DOCUMENT RESUME

ED 038 034 EM 007 932

AUTHOR	Feurzeig, W.; And Others
TITLE	Programming-Languages as a Conceptual Framework for Teaching Mathematics. Final Report on the First Fifteen Months of the LOGO Project.
INSTITUTION	Bolt Beranek and Newman, Inc., Cambridge, Mass.
SPONS AGENCY	National Science Foundation, Washington, D.C.
REPORT NO	R-1889
PUB DATE	30 Nov 69
NOTE	329p.
EDRS PRICE	EDRS Price MF-$1.25 HC-$16.55
DESCRIPTORS	*Computer Assisted Instruction, Educational Research, Instructional Technology, *Mathematics Instruction, Program Evaluation, Programing, *Programing Languages
IDENTIFIERS	LOGO

ABSTRACT

A new mathematics curriculum was used in this study which depended fundamentally on the use of computers and programing for presentation. The main part of the research was done with seventh grade children utilizing a programing language, LOGO, specifically designed for the teaching of mathematics. An investigation was also conducted with a group of second and third graders. After a brief exposition of the LOGO language, the two teaching activities are described in some detail, including many examples of the classroom and laboratory materials used. The report begins with a discussion of the reasons that mathematics instruction is so difficult, and states the underlying issues that have dictated the kind of approach taken here. Following the descriptive material on the teaching experiments is a discussion of the results, including some evaluation of the year's work and of the project. A detailed description of the LOGO programing language and system is appended. (Author/JY)

BOLT BERANEK AND NEWMAN INC

CONSULTING • DEVELOPMENT • RESEARCH

ED038034

Report No. 1889

30 November 1969

PROGRAMMING-LANGUAGES AS A CONCEPTUAL FRAMEWORK FOR TEACHING MATHEMATICS

W. Feurzeig
S. Papert
M. Bloom
R. Grant
C. Solomon

Submitted to:

National Science Foundation
Office of Computing Activities
1800 G Street, NW
Washington, D. C. 20550

EM007932

CAMBRIDGE NEW YORK CHICAGO LOS ANGELES

ED038034

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY.

Programming-Languages as a Conceptual Framework for Teaching Mathematics

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

W. Feurzeig
S. Papert
M. Bloom
R. Grant
C. Solomon

Final Report on the first fifteen
months of the LOGO Project

30 November 1969

Submitted to:

National Science Foundation
Office of Computing Activities
1800 G Street, NW
Washington, D. C. 20550

Contract NSF-C 558

TABLE OF CONTENTS

Page

1. **Foreword** 1

2. **The Research Problem** 3

2.1 The Contribution of LOGO 7

2.2 Introduction to the LOGO Language 16

2.3 The Project 28

3. **Elementary Teaching Investigation** 30

3.1 Overview 31

3.2 The Children's Work 34

3.3 The Lesson Materials 67

3.4 The Games 111

4. **Junior High School Teaching Experiment** 124

4.1 Design and Operation of the Course 124

4.2 LOGO Teaching Materials 126

Course Outline 126

Formal Elements 128

Heuristic Work 143

Early Projects 159

4.3 Algebra Materials 177

Sequences and Oscillators 178

Guessing and Strategy 193

Arithmetic Operations 215

Algebra Teaching Sequence 225

TABLE OF CONTENTS (continued)

Page

4.4 Evaluation 249

Achievement Test Results 249

Student Performance-Level Changes 256

Comments of Evaluators 259

Conclusions of Project Staff 271

Appendix: A Description of the LOGO Language and System

1. The LOGO Language 274

2. The LOGO System 306

3. Summary of LOGO Operations, Commands, Special Names, and Abbreviations 322

1. Foreword

This is a report of research and teaching directed toward the development of a new mathematics curriculum whose presentation depends *fundamentally* on the use of computers and programming. The work was centered mainly on a mathematics teaching experiment with seventh grade children utilizing a programming language, LOGO, specifically designed for the teaching of mathematics. We also conducted an investigation of the use of LOGO in teaching much younger children -- a group of second and third graders.

After a brief exposition of the LOGO language, the two teaching activities are described in some detail, including many examples of the classroom and laboratory materials used. The report begins with a discussion of the reasons why the learning and teaching of mathematics are so difficult, and states the underlying issues that have dictated the kind of approach undertaken here. Following the descriptive material on the teaching experiments is a discussion of the results including some evaluations of the year's work and of the project. A detailed description of the LOGO programming language and system is appended.

The seventh grade class was taught by Mrs. Marjorie Bloom from September 1968 through December 1968, and jointly by Miss Cynthia Solomon and Dr. Seymour Papert from January 1969 through June 1969. Dr. Papert, Professor of Applied Mathematics at Massachusetts Institute of Technology, was a consultant to Bolt Beranek and Newman on this project. During the latter period, Mrs. Bloom taught the group of second and third grade children.

We did not begin the teaching with a large body of previously developed classroom materials. These had to be created concurrently with the teaching as the courses progressed. The dynamic aspects of this day-to-day work helped assure that the content and presentation were adapted to the current needs of the children and were responsive to their difficulties, some of which we had not anticipated. The dedication, resources, and hard work shown by Mrs. Bloom, Miss Solomon, and Dr. Papert in responding to these challenges were exceptional.

The original research leading to the design of LOGO was supported by the U. S. Office of Naval Research. Dr. Papert, Dr. Daniel G. Bobrow, and Wallace Feurzeig designed the original version of the language. LOGO was first implemented by Dr. Bobrow and in an extended version by Richard Grant and Frank Frazier. The work of programming and maintaining the LOGO system for use in this project was initiated by Mr. Charles R. Morgan, now Chairman of the Department of Mathematics, Gordon College, Wenham, Mass., and was continued by Mr. Grant who also contributed to the design of the system as it evolved throughout the year.

Mr. Feurzeig coordinated the research design and implementation. The philosophical and pedagogical point of view adapted for the project was largely due to Dr. Papert. Miss Solomon contributed to the development and use of the language.

The work of installing and maintaining the computer terminals in the schools was done by Mr. Paul Wexelblat. Mr. Wexelblat and Mr. Grant were co-teachers of the computer club, an auxiliary activity at the junior high school.

The Muzzey Junior High School administration, particularly Mr. Santo Marino, Principal, and Mr. David Terry, Assistant Principal, were especially cooperative in providing a congenial classroom site for the project. Similarly, the Emerson School, particularly Mr. Donald Welch, Principal, provided a cordial base for the elementary teaching investigation.

This report was prepared by Mr. Feurzeig. Mr. Grant contributed to the writing and editing. Mrs. Frieda Ployer provided valuable critical review. Formatting, drawing, and final typing were done by Miss Pearl Stockwell.

2. The Research Problem

There is an old saying among mathematicians that there is no known theorem which cannot be made transparently clear to a high school student of average intelligence in a reasonable period of time (hours or months, not lifetimes). Yet few high school students acquire an understanding of even the simplest theorems and, for most students, the formal methods of mathematics remain forever mysterious, artificial, poorly motivated, and very obscurely related to intuitive thinking.

The relation of school children to mathematics remains deeply puzzling after more than a decade of wide-scale experiment in the classroom and in the cognitive laboratory. The extent of the puzzle is often obscured by popular prejudices about mathematics and about children. For if one asks: "why cannot *every* child learn algebra in a week?" the answer is likely to be influenced by glib thoughts like "math is difficult" and "*no one* learns *that* fast." But the question is a serious one and requires us to ask: wherein is mathematics difficult? What rational analysis convinces us there is that much to learn? Some things can be learned in ten minutes; why do children need so very long to understand equations or the manipulation of negative numbers?

Failure to obtain quick learning in classrooms is not in itself an indication of the quantity or difficulty of what has to be learned. It can be an indication that the teaching method is inadequate. In fact, the guiding thought of the following pages is the conjecture that current teaching does not even attempt to identify and teach those skills, concepts, and facts most needed by the child. This applies as much, sometimes more so, to most of the trends called "New Math" as to really traditional mathematics teaching.

To emphasize the sharpness of the position developed below, the following analogy may be useful. Most schools teach singing in a way that shows the Grant phenomenon: children are given the instruction "sing!" - those who can, do, those who cannot, become listeners. An observer watching the class over the whole year would see a great deal of teaching: the children who know how to sing learn new songs, new tunes, even new techniques of singing. But, all this teaching presupposes that *the really important learning has taken place elsewhere*.

Does this picture apply to our mathematics teaching? Do we give children the instruction "think!" without even telling them *how* to think. Does it all consist of teaching delightful mathematical songs to those who are lucky enough to have picked up the skill of mathematical thinking?

These questions open a theoretical dispute about which very sharp views are held. Can one *tell* children how to think? Some people believe very strongly that one certainly cannot, indeed that one cannot even tell them how to do arithmetic. For example, R. Davis,* one of the most serious innovators of active and creative kinds of mathematical activity for children, says:

> There is another reason for using "discovery": in point of fact you usually *cannot* "tell" the student what to do. You and he do not share a sufficiently precise meta-language.

Insofar as he is describing the status quo, Davis is certainly right. Occupants of present-day mathematics classrooms do indeed lack a "sufficiently precise meta-language". Students are accustomed to using language and logic in the context of a sympathetic listener who makes reasonable interpretations of their statements, and is tolerant of the gaps in their arguments. The formal mode of thinking imposed in the mathematics class seems arbitrary and unreasonable to them.

The low degree of mathematical articulation - amongst teachers as much as children - is at least partly the result of the following factors:

(a) The complete absence of a standard teachable terminology to discuss the *heuristic aspects* of mathematical activity concerned with the art of solving problems. In fact, these aspects (as opposed to *formal* ones) are scarcely recognized by official mathematics as worthy of study and teaching.†

---

*Davis, Robert B., *The Madison Project's Approach to a Theory of Instruction*, Journal of Research in Science Teaching, Vol. II, pp. 146-162, 1964.

†For further elaboration of this concept see the well-known works of Polya. See also Minsky, M. L., *Semantic Information Processing*, M.I.T. Press, 1969.

(b) In particular the relation of formal detail to global planning in working a problem is not clearly made in any standard treatment of elementary mathematics. Formal rigor is seldom properly understood by teachers as a working tool (rather than a fund of intellectual ritual).

(c) The traditional curriculum content is poor in that it seldom provides many examples of the same phenomenon. As a result, children are not familiar *from experience* with such basic processes as generalizing a method, extending the domain of an operation, and so on.

(d) As a consequence of the previous point, the possibilities of "discovery" are greatly impoverished - the child who did not make, but did understand, any particular discovery has little chance of using his understanding to try his hand at a related problem.

Indeed, we might summarize all these points by saying that school children have been deprived of the opportunity of actually *doing mathematics* in any sense even thinly related to the working activity of mathematicians. Thus, it is not surprising that children resist, that they seldom carry over their training in formal manipulation into less formal situations, and that they so often slip back into loose and uncontrolled thinking when faced with problems such as "word problems" in algebra that do not have obvious mechanical solutions.

To remedy, or even to study, this situation, one would like to find areas of mathematical work in which students would impose the need for precise articulation on themselves. We believe that such areas can be created by appropriate instruction in the use of computers and programming languages. The purpose of this research has been to investigate the teaching of mathematics in terms of a "sufficiently precise meta-language," the programming language LOGO, and to explore means of using it as the foundation and framework for a mathematics curriculum.

2.1 The Contribution of LOGO

Appropriate teaching with a suitable programming language can contribute to mathematics education in several ways.

(1) Programming facilitates the acquisition of rigorous thinking and expression. Children impose the need for precise statement on themselves through attempting to make the computer understand and perform their algorithms.

(2) Programming can be used to give students very *specific* insights into a number of key concepts. Ideas such as variable and function remain, to say the least, obscure for many high school students. Indeed, college students often have trouble with the many roles of the "x" in algebra: sometimes it appears to be a *number*, sometimes a subtly different kind of object called a *variable*, and on other occasions it is to be treated as a *function*. We contend that the difficulty stems less from the intrinsic intellectual subtlety or complexity of these distinctions than from their ethereal relation to anything in the real and familiar world. Moreover, it is possible to fumble one's way through an algebra course without ever facing these issues squarely. In programming, the distinctions arise concretely; they must be faced; and the physical nature of the machine provides a more earthy reference than can any abstract work. These

ideas should be easier in this context and our experience is that they are.

(3) Programming provides highly motivated models for all the principal heuristic concepts, for example:

> It lends itself perfectly to discussion of the relation of formal procedures to intuitive understanding of problems. It provides a wealth of examples for heuristic precepts such as "formulate a plan", "separate the difficulties", "find a related problem", etc. Thus, it provides a natural context to concretize the approach to teaching associated with the name of George Polya.
>
> It provides a sense of completely formal methods and what their purpose is. It gives the child a chance to learn to distinguish situations where complete formal rigor is necessary from those where looser thinking is appropriate.
>
> In particular, it provides models for the contrast between the global planning of an attack on a problem and the formal detail of an elaborated solution. In the context of programming, the concept of sub-problem or sub-goal emerges crisply. It is at least highly plausible that pupils who have acquired very early the habit of organizing their approach to a mathematical problem will be better able to develop systematic habits of thought in the more murky areas of problem-solving they will have to meet later, in school and elsewhere.
>
> The concrete form of the program and the interactive aspect of the machine allow "debugging" of errors to be identified as a definite, constructive, and plannable activity. The

programming concept of a "bug" as a definite, concrete, existent entity to be hunted, caught, and tamed or killed is a valuable heuristic idea.

(4) By enlarging the scope of applications, it allows every problem to be embedded in a large population of related problems of all degrees of difficulty, for example:

Through programming, mathematical induction can be presented and generalized by its relation to recursion. An example of this kind of presentation is shown in Section 2.2. The examples given in Section 4.3 show how we have learned to present recursion itself as related to the general heuristics of planning.

The extension of an operation to a larger domain becomes an everyday activity. The newer mathematics texts do emphasize the extension of addition, for example, to successively more general kinds of numbers (integers → rationals → reals). But the phenomenon is obscured for children by its isolation and by the fact that children already know how to add real numbers.

Generalizing this, *generalization* becomes an activity undertaken routinely by the children.

Functions become familiar things one invents oneself to serve real purposes. We have seen children invent as many new functions in a week as they would otherwise learn (by rote!) in their whole career. More importantly, they use these functions as building blocks for constructing more complex functions which often are elements of still larger constructs -- very much in the way mathematicians use propositions to prove theorems and use these theorems to prove more complex theorems.

(5) The use of computers and programming languages is also relevant to what is perhaps the most difficult aspect of mathematics for a teacher: helping the student strive for self-consciousness and literacy about *the process of solving problems*. High school students can seldom *say anything* about how they worked towards the solution of a problem. They lack the habit of discussing such things and they lack the language necessary to do so. A programming language provides a vocabulary and a set of experiences for discussing mathematical concepts and problems. Programs are more *discussable* than traditional mathematical activities: one can talk about their structure, one can talk about their development, their relation to one another, and to the original problem.

(6) A related point is that the computer can be used as a *mathematical laboratory* to foster an experimental approach toward solving problems. Programming could, in principle, be taught as an abstract mathematical topic without using or, indeed, even mentioning computers. Presented in that spirit, the material would retain some of the pedagogical virtues that motivate our interest in it. But an *essential* aspect would be lost. The use of a computer has the major merit of turning a programming language into an active instrument to control an outside reality. The most immediate effect of using a computer is that explicit and precise statement is no longer imposed by the arbitrary edict of a teacher but by the obvious necessities of making the computer do one's bidding. Since students learn to write programs by

experience and experiment, it is appropriate to use the term mathematical laboratory for the practical phases of the instruction.

The reason that a laboratory is not traditionally used in mathematical study is not that it would be less valuable there than in biology, chemistry, or physics; rather, the idea of a mathematical experiment was, until recently, unrealizable, and barely conceivable, except in very special or superficial senses. How could a person set in motion a sequence of mathematical events or a mathematical process, and then see its effects unfold? Using a computer with an appropriate programming language adds this extra dimension to mathematical experience; the important contribution of the computer is a new and powerful operational universe for mathematical experiments.

(7) Finally, the richness of non-numerical examples open to programming can be exploited to enlarge the cultural base of the mathematics course by bringing it into contact with physical and biological science, language study, geography, economics, and other subjects.

Thus, our interest is not to teach programming as an auxiliary topic, but to explore means of using it as a *foundation* for an integrated course in mathematics. This concept of programming is distinct from the already familiar and valuable ones of teaching computer programming as a practical skill in its own right or for use in special courses in numerical applications, applied mathematics, computational methods, and the like.

In almost all educational uses of programming languages to date, the particular languages employed were not originally designed for teaching. Most of the languages used, including FORTRAN, APL, and JOSS (which has many dialects such as TELCOMP, CAL, and PILL), were originally designed for computational applications in mathematics, science, and engineering. Some of these were subsequently modified, usually in minor ways, to adapt them for use in teaching. A few languages, notably BASIC, were designed for teaching programming as a skill, and for providing students with experience in its use as a "problem-solving" tool. Educationally beneficial applications of many kinds have been made through such use of these languages.

We now present the considerations that led us to create the programming language LOGO. The introduction of yet another language clearly deserves critical examination, particularly since several existing languages appear to be suitable for teaching mathematics. The JOSS languages, for example, have been described as exceptionally well-suited for use in mathematical work: it has been pointed out that "all that one needs to know to start writing JOSS programs, almost instantly with very little preparation, is algebra." That observation is well-taken but it points up the problem: most students leave school *without having learned* algebra -- it is precisely for the purpose of teaching mathematics, rather than assuming that children already know it, that we want to use a programming language. (We do not want to tell them "Sing!" before we teach them how.)

It might reasonably be argued that this difficulty is only apparent and that existing languages *could* be used to teach arithmetic and algebra. Indeed, starting with this objective and the requisite point of view, one could consider using JOSS or BASIC as a foundation and framework for mathematics. But it would not be easy -- these languages were not designed to teach the most elementary (and often the most difficult) concepts and skills, and constructive methods of extending them.

For these purposes, existing languages usually have too much mathematical machinery built in: to use JOSS and most of the others normally requires a knowledge of decimal notation and scientific representation (floating point numbers, exponential numbers) and some familiarity with the use of arithmetic expressions. At the same time, most of these languages are not well-suited for expressing formal or non-numerical procedures.

These, however, are negative considerations. More to the point are the positive reasons which motivated us. Earlier experiences with the use of programming in elementary and secondary mathematics teaching convinced us of the need for a language, i.e., LOGO, with the following characteristics.

(1) It should be accessible to young children and others who have not acquired the elements of mathematical thinking. The only prerequisites for using it should be an acquaintance with the counting numbers and the ability to read at about second-grade level.

(2) It should be transparently direct, natural-seeming, and easy to use for expressing procedures for simple tasks like many non-numerical problems already familiar to children.

To meet these two requirements, the language should be without difficult technical features like those found in traditional programming languages (e.g., loops, counters, array declarations, multiple mode arithmetic, etc.).

(3) It should be organized to facilitate the extension and generalization of simple mathematical algorithms to more advanced and powerful ones. For example, the most primitive numerical operations in LOGO are centered on integer arithmetic and can only be used for counting or for adding and subtracting integers. But students can write LOGO procedures for expanding these arithmetic operations into mathematically rich and advanced algorithms in arithmetic, algebra, and higher mathematics with appropriate ease.

(4) The structure of the language should embody mathematically important concepts and foster the development of a constructive point of view about mathematical work.

Solving a mathematical problem is a process of construction. The activity of programming a computer is uniquely well suited to transmitting this idea. The image we would like to convey could, roughly speaking, be described thus: A solution to a problem is to be built according to a preconceived, but modifiable, *plan*, out of *parts* which might also be used in building other solutions to the same or other problems. A partial, or incorrect, solution is a useful object; it can be extended or fixed, and then incorporated into a large structure. This image is mirrored in the activity of writing LOGO programs. Using procedures as building blocks for other procedures is standard and natural in LOGO programming. The use of functionally separable and nameable procedures composed of functionally separable and nameable parts, coupled with the use of recursion, makes the development of constructive mathematical methods meaningful and teachable.

Students construct LOGO procedures from the very beginning, as they are introduced to the language. They start with non-numerical procedures with which they are all familiar. Good examples are translating English into Pig Latin, making and breaking secret codes (e.g., substitution ciphers), a variety of word games (finding words contained in words, writing words backwards, etc.), question-answering and guessing games (Twenty Questions, Buzz, etc.). There are many problems of this sort which children already know and like. The student thinks at first that he understands such problems perfectly because, with a little prodding, he can give a loose verbal description of his procedure. But he finds it difficult to make this description precise and general partly for lack of formal habits and partly for lack of a suitably expressive language. The initial value of using LOGO becomes apparent when the student attempts to make the computer perform his procedure. At this point the process of transforming loose verbal descriptions into precise formal ones becomes possible and, in this context, seems natural and enjoyable to children.

An understanding, or even a clear appreciation, of these points is impossible without a brief introduction to the LOGO language. The presentation that follows is not a complete description of LOGO. Its purpose is merely to give a sense of the spirit and structure of LOGO programming. Some pedagogically important operations and commands are not even included here, for example, the REQUEST operation which makes possible the writing of interactive procedures. These are introduced in the body of the report along with the features of the LOGO system having to do with editing, correcting program errors, and filing programs for subsequent retrieval. A comprehensive description of the LOGO language and system is included as an appendix to the report.

2.2 Introduction to the LOGO Language

LOGO is a language for expressing formal procedures. LOGO procedures are written along lines similar to recipes in cooking. A procedure, like a recipe to bake a cake, has a name; it usually has ingredients, maybe several, but maybe none (these are called its inputs); and it has a sequence of instructions telling how to operate upon its inputs (and upon the things made from them along the way) to produce the desired effect or to make a new thing (this is called its output).

To illustrate, we define a procedure for doubling a number. We begin by choosing a word for the name of the procedure - let's choose the word DOUBLE in this case. Next we choose names for the inputs - in this case there is a single input, which we'll call NUMBER. So, the title of the procedure is

```
TO DOUBLE /NUMBER/
```

(like to boil an egg). Note the slash marks around NUMBER -- slashes are used to demarcate names of things; names for procedures like DOUBLE and for already-built-in LOGO instructions are written without any marks around them.

When we give LOGO the command DOUBLE 5 we want the teletype to respond 10; when we say DOUBLE 9999 we want the response 19998. So now we proscribe the instructions for performing this. One instruction suffices:

```
PRINT SUM OF /NUMBER/ AND /NUMBER/.
```

This instruction is composed of two elementary (i.e., originally built-in) instructions -- PRINT and SUM.

PRINT is a command which needs one input (this can be any LOGO thing - a number or some other alphanumeric word or a sentence comprising several words). Its effect is to cause the teletype to print its input. Thus, PRINT 752 causes the teletype to print 752; PRINT "GOOD MORNING" causes the teletype to print GOOD MORNING. (Quotation marks are used to indicate LOGO things that stand for themselves. Since integers always stand for themselves in LOGO, they do not need to be quoted.)

SUM is an operation which needs two inputs (these must be integers). Its output is their sum. Thus, SUM OF 3 AND 2 has the output 5. The LOGO instruction:

```
PRINT SUM OF 3 AND 2
```

causes the teletype to print the LOGO thing which is the output of SUM OF 3 AND 2, i.e., 5.

The entire procedure definition is:

```
TO DOUBLE /NUMBER/
1 PRINT SUM OF /NUMBER/ AND /NUMBER/
END
```

where the integer 1 is used to label the instruction line (in this case there is only one line, but procedures often have several lines of instructions), and END marks the end of the definition. When this completed definition is typed in, LOGO acknowledges by responding: DOUBLE DEFINED. From that point on, the procedure DOUBLE can be used as if it had always been part of LOGO, just like PRINT and SUM. The new procedure is used by typing:

```
DOUBLE 2
```

The machine responds with the answer

```
4
DOUBLE 4
8
```

(We underscore the student's or teacher's typing in these and the following examples to distinguish them from LOGO's responses.)

Of course, rather than write a procedure for something as simple as DOUBLE, we can accomplish the same thing merely by writing:

```
PRINT SUM OF 2 AND 2 or
PRINT SUM OF 4 AND 4, etc.
```

Using the procedure requires less writing however, and we might want to use it a great deal.

But, if we want to use it in a compound instruction chain like:

```
DOUBLE DOUBLE 4
```

where we expect the result to be 16, DOUBLE will not work properly: it will print 8 and then it will print an error message. The difficulty is that DOUBLE, as written, does not provide its result as an *output* to another procedure; it merely prints its result out on the teletype. Procedures (and built-in instructions) that have an output are called *operations* to distinguish them from *commands* which have no output. We can change DOUBLE to a procedure that defines an operation, as follows.

```
TO DOUBLE /NUMBER
1 OUTPUT SUM OF /NUMBER/ AND /NUMBER/
END
```

Here, the elementary command OUTPUT is used in place of PRINT. To use this new DOUBLE operation we write, with an external PRINT command,

```
PRINT DOUBLE OF 2
4
PRINT DOUBLE OF 4
8
PRINT DOUBLE OF (DOUBLE OF 4)
16
etc.
```

The use of parentheses is optional. In the last example DOUBLE OF 4 produces the output 8 for use as the input to the first-written DOUBLE, whose output is therefore 16.

There are a relatively small number of elementary operations and commands in LOGO. An operation which is analogous to the operation SUM for integers is the operation WORD for alphanumeric words. Thus, PRINT WORD OF "SUN" AND "ABC" will cause the LOGO word SUNABC to be printed. PRINT WORD OF WORD OF "AB" AND "123" AND "GO" will cause the word AB123GO to be printed. A procedure defining an operation on words, analogous to DOUBLE on numbers, can be written as follows:

```
TO DUBBLE /WD/
1 OUTPUT WORD OF /WD/ AND /WD/
END
DUBBLE DEFINED         (LOGO acknowledges)

PRINT DUBBLE OF "GO"
GOGO
PRINT DUBBLE OF DUBBLE OF "LA"
LALALALA
etc.
```

Two operations closely related to SUM and WORD are DIFFERENCE (or its abbreviation DIFF) and SENTENCE. Their use is illustrated by:

```
PRINT DIFF OF 3 AND 1
2
PRINT DIFF OF 1 AND 3
-2
PRINT SENTENCE OF "SUN" AND "STARS"
SUN STARS
```

```
PRINT SENTENCE OF "THIS IS" AND "GOOD"
THIS IS GOOD
```

The operations SUM and WORD are used to put things together. There also are LOGO operations of the opposite kind, for extracting components of things. Four such operations FIRST, LAST, BUTFIRST, and BUTLAST work as follows:

```
PRINT FIRST OF "BOX"
B
PRINT LAST OF "BOX"
X
PRINT BUTFIRST OF "BOX"
OX
PRINT BUTLAST OF "BOX"
BO
PRINT BUTFIRST OF "I LIKE YOU"
LIKE YOU
PRINT BUTLAST OF "I LIKE YOU"
I LIKE
PRINT BUTFIRST OF BUTLAST OF "ABCD"
BC
```

Note that BUTFIRST means *all but the first* letter of the word (or word of a sentence) and BUTLAST means *all but the last* letter (or word), and that these are operations, thus they can be chained together.

Some elementary LOGO operations have no inputs. Examples are CLOCK and RANDOM. The use of these is illustrated by:

```
PRINT CLOCK
123
PRINT CLOCK
125
```

Here we see that 123 seconds had elapsed between the time the student started working and the time that the first of the two PRINT commands was performed, and that 2 seconds more elapsed before the second PRINT was performed.

```
PRINT RANDOM
7
PRINT RANDOM
4
```

RANDOM has as its output a single digit number chosen randomly from a uniform distribution. To make a two digit random number we write:

```
PRINT WORD OF RANDOM AND RANDOM
36
```

Two basic acts in procedures are making new LOGO things and testing them to see whether they satisfy some condition, such as a stop rule. To tell LOGO that we want to make a new LOGO thing, we type the command MAKE. LOGO responds by asking us first for the name we want to give the new thing and then for the thing we want to make, i.e., for a LOGO expression for the new thing. Thus, if we want to make a sentence named "GOODIES" out of some words for foods we like, we can write:

```
MAKE
    NAME:  "GOODIES"
    THING:  "APPLES BUNS CAKES PIES"
```

If we then type
`PRINT THING OF "GOODIES"`, LOGO responds

```
APPLES BUNS CAKES PIES
```

(If we had typed instead `PRINT "GOODIES"`, LOGO would have responded GOODIES.)

A shorthand way of writing THING OF (to indicate that we mean the thing being named rather than the name) is by using slashes instead of quotation marks. Thus,

```
PRINT /GOODIES/
```

means the same as PRINT THING OF "GOODIES" and so produces the same response,

```
APPLES BUNS CAKES PIES.
```

Similarly,

```
PRINT FIRST OF BUTFIRST OF /GOODIES/
```

causes LOGO to print

```
BUNS.
```

To test whether or not a LOGO thing satisfies a specified condition, we introduce the concept of predicate, i.e., an operation whose possible outputs are "TRUE" and "FALSE". The identity operation IS is one of the elementary LOGO predicates. IS takes two inputs and has the output "TRUE", if these inputs are the same, and the output "FALSE", if they are different. Thus,

```
PRINT IS 2 SUM OF 1 AND 1
TRUE
PRINT IS 2 1
FALSE
```

Other elementary predicates include GREATERP, NUMBERP, and WORDP.

```
PRINT GREATERP OF 2 AND 1
TRUE                          (because 2 is greater than 1)
PRINT GREATERP OF 1 AND 2
FALSE                         (because 1 is not greater than 2)
PRINT NUMBERP OF "ONE"
FALSE                         (because "ONE" is not a number)
PRINT NUMBERP OF 1
TRUE                          (because 1 is a number)
```

```
PRINT WORDP OF "ONE"
TRUE                             (because "ONE" is a word)
PRINT WORDP OF "THIS WORD"
FALSE          (because "THIS WORD" is a sentence, not a word)
```

The command TEST, along with its associated commands IF TRUE and IF FALSE, is used with a predicate as in the following examples.

```
TEST IS 2 2
IF TRUE PRINT "GOOD"
```

causes the machine to print GOOD. On the other hand, when the instructions

```
TEST IS 2 2
IF FALSE PRINT "BAD"
```

are performed, nothing will be printed.

The use of the commands MAKE and TEST is illustrated in the following procedures for printing random numbers.

```
TO NUMBER
1 PRINT RANDOM
END
```

This procedure is used by typing:

```
NUMBER
```

The machine responds with a number

```
8
NUMBER
5
```

etc.

The repetitive act of typing NUMBER is easily mechanized by writing a new procedure to do just this, i.e.,

```
TO SPEW
1 NUMBER
2 SPEW
END
```

We have incorporated into SPEW the instruction to perform another procedure, NUMBER, and then the instruction to SPEW, i.e., to do the same again. So when we type SPEW, we obtain an endless sequence of numbers:

```
SPEW
7
3
0
9
.
.
.
```

As well as using another procedure, NUMBER, SPEW also uses itself -- it is a simple example of a recursively defined procedure. To modify SPEW so as to produce a definite number of random digits, we introduce a new actor on the problem scene: the number of times we still have to SPEW. We name this actor "TIMES" and write:

```
TO SPEW /TIMES/
1 TEST IS /TIMES/ 0
2 IF TRUE STOP
3 PRINT RANDOM
4 MAKE
    NAME:  "NEWTIMES"
    THING:  DIFFERENCE OF /TIMES/ AND 1
5 SPEW /NEWTIMES/
END
```

The use of this new SPEW procedure is illustrated by:

```
SPEW 4
0
3
7
9
SPEW 4
2
8
1
7
```

A similar non-numerical recursive procedure, TRIANGLE, was invented by a child. It is defined as follows:

```
TO TRIANGLE /WORD/
1 TEST IS /WORD/ /EMPTY/
2 IF TRUE STOP
3 PRINT /WORD/
4 MAKE
    NAME:  "NEWWORD"
    THING:  BUTFIRST OF /WORD/
5 TRIANGLE /NEWWORD/
END
```

(/EMPTY/ denotes the empty thing in LOGO, i.e., the word with no letters)

To use TRIANGLE we write -

```
TRIANGLE "CIRCLE"
CIRCLE
IRCLE
RCLE
CLE
LE
E
```

The factorial function is an illustration of a deeper recursive procedure closely related to the principle of "mathematical induction". The definition of the factorial function is

FACTORIAL(1) = 1
FACTORIAL(N) = N x FACTORIAL(N-1), N > 1

In LOGO we write -

```
TO FACTORIAL /N/
1 TEST IS /N/ 1
2 IF TRUE OUTPUT 1
3 MAKE "N-1" DIFF OF /N/ AND 1
4 OUTPUT PRODUCT OF /N/ AND FACTORIAL OF /N-1/
END
```

To use FACTORIAL we write -

```
PRINT FACTORIAL OF 7
5040
```

Note in the above procedure the use of a PRODUCT operation for integer multiplication and the use of the two-input form of the MAKE command.

A similar non-numerical procedure for reversing the order of the letters in a word (i.e., writing it backwards) is:

```
TO REVERSE /WORD/
1 TEST IS COUNT OF /WORD/ 1
2 IF TRUE OUTPUT /WORD/
3 MAKE "NEWWORD" BUTLAST OF /WORD/
4 OUTPUT WORD OF LAST OF /WORD/ AND REVERSE OF /NEWWORD/
END
```

To use REVERSE we write -

```
PRINT REVERSE OF "ELEPHANT"
TNAHPELE
PRINT REVERSE OF FACTORIAL OF 7
0405
```

Note in the above procedure the use of the COUNT operation - COUNT of a word (sentence) is the number of letters (words) in the word (sentence). Note also that the name /WORD/ is as distinct from the operation WORD as it is from the literal word "WORD".

In LOGO the principle of mathematical induction is embedded in a more general class of recursive principles. These can be systematically investigated in a range of cases of increasing difficulty starting from the trivial recursion in the earlier SPEW procedure, proceeding to simple recursions like that in TRIANGLE through deeper examples as in FACTORIAL and REVERSE, and then beyond. The study of recursive procedures can provide a valuable approach to understanding the formal ideas underlying mathematical reasoning.

Experience in writing LOGO procedures is equally valuable in teaching the heuristic aspects of mathematical work. Such experience is fostered by projects that involve writing several procedures to function together as a single program. This kind of activity was an important part of the seventh grade classroom work during the year. Several examples of such projects are shown and discussed in Part 4.3.

Some changes were made in the nomenclature of LOGO at the end of the school year for pedagogic and mathematical reasons. Thus, in the programs encountered in the body of the report, the reader should note that RETURN is a synonym for OUTPUT, and CALL is a synonym for MAKE (here the order of the inputs for "NAME" and "THING" is reversed). Further, TEST was not used. Predicates (e.g., IS) stood alone and IF YES, IF NO were used in place of IF TRUE, IF FALSE.

2.3 The Project

This project concerns the use of LOGO as a framework for teaching mathematics. Specifically, our study explored that idea in the following ways.

(1) Students. We deliberately chose to work with a small class of "*average*" seventh-grade students. (Ten of the students were in the middle mathematics track of the school's five-track system. The other two had a slightly higher placement.) We chose a small class to facilitate more intensive study of individual children and to permit sufficient amount of individual student use of computer time (there were six computer terminals in one classroom). The school, Muzzey Junior High School in Lexington, Mass., was chosen mainly because of the relatively long class period - a full hour session, four days a week - which gave us some extra freedom in scheduling the students' time between classroom discussion and laboratory work at the terminals.

(2) Subject. Our goal was to give the students an introduction to high school algebra, which they normally would not have studied before ninth grade. (We managed to get a good start on this despite having to take more time than we anticipated in teaching LOGO itself.)

(3) Presentation. The mathematical material in the course was introduced and developed *wholly* and *entirely* in terms of LOGO programs. (This included the classroom teaching of all the arithmetic and algebra, not just material assigned for working out at the computer terminals.)

The major object of this work, the exploratory development of a new curriculum, was to test the feasibility of the underlying ideas about content and presentation by putting them in tangible form and trying them out in the classroom. The main activity was the junior high school teaching experiment. During the last half of the school year we expanded the effort by starting up a parallel activity -- teaching LOGO to a small group of second- and third-grade students. Because the work with elementary school children introduces LOGO with particular ease, we present it first.

3. Elementary Teaching Investigation

The purpose of this part of the work was to gain an understanding of the problems of teaching formal skills to very young children. An appropriate foundation for learning formal ways of thinking at an early age could have a profound impact on subsequent intellectual development. We thought it plausible that LOGO could be taught to second- or third-grade children as a starting point.

Taken at face value, Piaget, and most other serious students of developmental psychology, must be read as casting important doubts on the feasibility (or even the advisability) of teaching LOGO to children of age eight or nine. Our confidence that such an experiment was worth pursuing was based on a careful consideration of the real content of Piaget's thesis and on the nature of its experimental validation. The two major points are the following:

(1) No serious controlled attempts have been made to teach what Piaget would call "formal thinking" at much earlier ages than it naturally develops. Indeed, we would argue that programming provides a uniquely powerful tool for this and so, by its very nature, invalidates any negative conclusions drawn from previous experiments.

and

(2) The apparent difficulties suggested by the psychologists (and, indeed, by common knowledge of children) apply unequally to different aspects of learning to program. Thus there is nothing in Piaget's writing to suggest that a seven-year-old child should have the slightest difficulty dealing with programs such as

```
TO MUMBLE /JUMBLE/
1  PRINT /JUMBLE/
2  MUMBLE /JUMBLE/
END
```

They do suggest that children of this age should find it very hard to debug the kind of program with a branching structure which makes it necessary to hold in mind a number of possible outcomes or to carry out "hypothetico-deductive" experiments to formulate a theory of what is wrong.

These two points together urge the quest for an area of programming in which one can find suitable problems to provide a motivated learning foundation for small children without going beyond the more "elementary" program forms. Once such a foundation is established, it will become possible to probe the true difficulties that face teaching formal skills to small children.

As a first step in this preliminary study, our goal was to determine whether or not some very young children could learn the elements of LOGO programming.

3.1 Overview

This work was conducted at the Emerson Elementary School in Newton, Massachusetts. We installed a single computer terminal there at the end of January 1969. The school chose the children who were to participate in the study. These comprised, for the most part, mathematically "average" children whose ages ranged from seven through nine, though there were some "underachievers" and one of the children was mathematically "brighter than average".

The children were taught by Mrs. Marjorie Bloom who had previously taught the elements of LOGO to the class of junior high school students. Mrs. Bloom is a professional teacher. She had virtually no previous experience with computers and programming prior to joining the project in July 1968.

The teaching of LOGO was done largely through a series of programmed lessons of a relatively open-ended sort. These were written by Mrs. Bloom in the LOGO language itself. They were used by each of the children in an interactive, conversational mode.

The kind of presentation used with the junior high school students, classroom teaching with associated individual work at the terminals, was not feasible here because of constraints on the childrens' time and schedule and the limitations posed from having only one computer terminal available. We were, however, interested in seeing that a presentation along these lines, properly monitored by a teacher, and augmented by some work in writing programs at the terminal, was feasible.

A narrative discussion of the work as documented in the teacher's daily log, and samples showing the childrens' use of all the teaching materials, follow in the next parts of the report. These give a good idea of what happened, i.e., of the childrens' progress and problems. Our main conclusions were as follows.

(1) Children of second and third grade level learn the *elements* of LOGO programming with ease.

(2) Most children at this level cannot, during such a short interval, learn to write or debug programs as complex as REVERSE (as in Part 2.2 above). Only one child was able,

within the four-month period, to deal freely with programs more complex than MUMBLE.

(3) Children of this age do acquire a meaningful understanding of concepts like variable, function, and formal procedure (though not those words) through their experience with LOGO.

(4) The children showed educational benefits of an extra-mathematical kind as side effects of the teaching. The most evident one was a striking improvement in reading rate for most children during this period. They acquired a technical vocabulary and learned to follow relatively sophisticated verbal instructions.

The remainder of this section includes the narrative description from the class log, transcripts of student runs of each of the series of programmed lessons, the LOGO teaching programs for a pair of typical lessons, and transcripts illustrating the games played by the children at the terminal from time to time throughout the course.

3.2 The Children's Work
- excerpts from the daily log

> Work with children at the Emerson School began January 29, 1969. Mrs. Bloom started with twelve children -- second, third, and fourth graders -- divided into four small classes. Because of snowstorms and a school holiday week, resulting in only five days of school in the first month, the group was reduced to two second graders and six third graders. Five of these children had individual instruction at the terminal for about 20 minutes a session, and three children worked together as a group, also for 20 minutes. The children had four sessions a week for eleven weeks from the first week of March to the end of May. In addition to the regular group of eight, an emotionally disturbed third grader participated for twelve 20-minute periods. From May 29 through June 13, instruction continued with only four children to see what additional progress might be made if they were given a little more time at the terminal. The maximum amount of instruction time for any child was about fifteen hours over the entire period.

We began each group with the HI procedure, a programmed greeting. It did not seem to bother anyone that the procedure typed out identical responses to each child. Some children typed in funny answers deliberately, such as RUTHANNEDUM for name and 500 for age.

```
←HI
HI, THERE!
WHAT'S YOUR NAME?
⁑RUTHANNEDUM                    (child's typed-in response)
HOW DO YOU DO, RUTHANNEDUM!
HOW OLD ARE YOU?
⁑500                            (child's typed-in response)
MY, YOU LOOK VERY GROWN UP FOR ONLY 500 YEARS OLD!
I HOPE YOU WILL HAVE A VERY GOOD TIME WITH ME, AND THAT
YOU WILL TRY A LOT OF FUN THINGS!
GOODBYE FOR NOW, RUTHANNEDUM!
←
```

The entire group was delighted and hysterical at the machine's answers. In two of the classes we went on to modify the HI procedure. The second graders added I LIKE YOU to the responses. The third graders wrote in YOU ARE VERY GOOD-LOOKING.

The next day, the two second graders started off playing the game FOUR-IN-A-ROW at the computer terminal. Mary Jaye lost her game, but Steven played to a draw. He seems to feel that this is tantamount to winning. Then, using an outline (on cardboard) and stickers, each child made his own copy of the teletype keyboard.

I showed the two classes comprising the oldest children (and Karen Coffey from the previous class, who didn't want to leave) the operation of two procedures, LAUGH and KEEPLAUGHING.

```
←LAUGH
HAHA

←KEEPLAUGHING
HAHA
HAHA
HAHA
HAHA
HAHA
HAHA
 : :
```

They were fascinated with KEEPLAUGHING although initially they showed some concern about how to make it stop. I teased them briefly before showing them the break key. Once they knew they could stop it whenever they wished, they were delighted to let it run on and on and were busy measuring the lengths of the paper which were printed out from each run. Since they enjoyed these two procedures so much, I encouraged them to write a CRY and a KEEPCRYING. Following the models of LAUGH and KEEPLAUGHING -

```
TO LAUGH
10 PRINT "HAHA"
END

TO KEEPLAUGHING
10 PRINT "HAHA"
20 KEEPLAUGHING
END
```

they wrote the following two procedures,

```
TO CRY
10 PRINT "BOOHOO"
END

TO KEEPCRYING
10 PRINT "BOOHOO"
20 KEEPCRYING
END
```

and tried them out.

```
←CRY
BOOHOO

←KEEPCRYING
BOOHOO
BOOHOO
BOOHOO
 :  :
```

The next day, after the success with CRY with the two older groups, I decided to try it with the second graders and the first class of third graders. They seemed to enjoy it just as much and they were equally successful with it. On Thursday I suggested a problem: write a procedure which will type something over and over again down the page. Each of the girls wrote a procedure which typed her first name down the page, and then Shawn wrote one which typed SHAWN MICHAEL DALEY down the page. The only difficulty Shawn had was spelling Michael.

Rosemarie brought in two procedures -- a KEEPROSEMARIEING and a KEEPMASTERMINDING. (She and Joan have decided to name the computer MASTER MIND.)

```
To KeepRosemarieing
1 Print "Rosemarie"
2 KeepRosemarieing
End
```

```
To KeepMasterminding
4 Print "Master Mind"
6 KeepMasterMinding
End
```

Rosemarie
Phillips

We typed them in and tried them.

```
←TO KEEPROSEMARIEING
>1 PRINT "ROSEMARIE"
>2 KEEPROSEMARIEING
>END
KEEPROSEMARIEING DEFINED
←KEEPROSEMARIEING
ROSEMARIE
ROSEMARIE
ROSEMARIE
ROSEMARIE
 :   :
```

```
←TO KEEPMASTERMINDING
>4 PRINT "MASTERMIND"
>6 KEEPMASTERMINDING
>END
KEEPMASTERMINDING DEFINED
←KEEPMASTERMINDING
MASTERMIND
MASTERMIND
MASTERMIND
MASTERMIND
 :   :
```

I introduced the operation of naming to the last class by example, without prior discussion. I simply typed in

```
CALL
   THING:  "DOG"
   NAME:  "SNOOPY"
```

and then we tried PRINT "SNOOPY", PRINT THING OF "SNOOPY", and PRINT /SNOOPY/.

On Wednesday, February 5, a second-grader, Steven, played two games of FOUR-IN-A-ROW as did Mary Jaye and Neil. While they were playing, Steven wrote a PRINTSTEVEN which printed the letters S,T,E,V,E, and N one under another. He wrote the procedure by himself. The only difficulty he had was in forgetting to demarcate the letters with quotation marks. Then he wanted to get his program to keep typing these letters down the page.

In the other classes I demonstrated and explained the operation COUNT for words and the children figured out what COUNT does for sentences. We went on to SUM and DIFFERENCE and they experimented with lots of numbers, checking the computer's accuracy.

Rosemarie taught the computer a simple poem. The procedure name was the title of the poem, and the instructions were to print the whole poem.

Because of snow emergencies and February vacation, class did not meet again until March 6. We continued with eight children - five working individually and three as a group.

(From this point, the children's daily work included short interactive computer-administered lessons. Typescripts from the first three lessons - TEACHPRINT, TEACHWORD, and TEACHSEN - are reproduced next to illustrate the early material and its use. The children's typing is underscored in these typescripts. The entire set of lessons is reproduced later on.)

```
←LESSON1

IN THIS LESSON THERE ARE SOME LINES TO READ. THE COMPUTER TYPES VERY
FAST. IT WILL STOP EVERY SO OFTEN SO THAT YOU WILL HAVE TIME TO READ.
WHEN YOU WANT IT TO GO ON, PRESS THE RETURN KEY. @

THIS MARK @ MEANS THAT THE COMPUTER HAS MORE TO TELL YOU. PRESS THE
RETURN KEY WHEN YOU ARE READY TO READ IT. @

THE COMPUTER UNDERSTANDS SOME SPECIAL COMMANDS. THE FIRST ONE WE WILL
TALK ABOUT IS -- PRINT. @

LET'S TELL THE COMPUTER TO PRINT A WORD. I WILL TYPE THE COMMAND. YOU
PRESS THE RETURN KEY WHEN I AM FINISHED.

PRINT "CABBAGE"
CABBAGE

NOW LET'S TELL THE COMPUTER TO PRINT A SENTENCE. YOU PRESS THE RETURN
KEY THIS TIME TOO.

PRINT "DO YOU LIKE THE BEATLES?"
DO YOU LIKE THE BEATLES?

NOTICE THAT THERE WERE QUOTATION (" ") MARKS AROUND THE EXACT WORDS THE
COMPUTER WAS ASKED TO PRINT. @

NOW TRY TO MAKE THE COMPUTER PRINT SOME OF YOUR OWN THINGS. @

DON'T WORRY ABOUT MAKING MISTAKES! WE ALL DO IT! THE COMPUTER WILL TRY
TO HELP YOU BY EXPLAINING WHAT WENT WRONG. @

HAVE FUN!
←
```

```
←LESSON2

REMEMBER -- THE MARK @ MEANS THAT I AM WAITING FOR YOU TO PRESS THE
RETURN KEY. @

AFTER YOU LEARN TO USE LOGO, YOU WILL BE ABLE TO TEACH THE COMPUTER TO
DO NEW THINGS. @

YOU WILL TEACH THE COMPUTER TO DO SOMETHING NEW BY WRITING A PROCEDURE.
@
```

```
A PROCEDURE IS LIKE A RECIPE. IT HAS ALL THE INGREDIENTS AND DIRECTIONS
 IN IT TO MAKE SOMETHING. @

A FEW PROCEDURES ARE ALREADY BUILT INTO LOGO. WE WILL TALK ABOUT THESE
BEFORE WE TALK ABOUT PROCEDURES THAT YOU WILL WRITE. @

THE PROCEDURE WE WILL LEARN ABOUT IN THIS LESSON IS CALLED -- WORD.
 LET'S SEE WHAT IT DOES. @

TYPE THESE WORDS --      PRINT WORD OF "UP" AND "DOWN"
*PRINT WORD OF "UP" AND "DOWN"
UPDOWN

HERE IS ANOTHER EXAMPLE OF WHAT -- WORD -- DOES.

TYPE THIS:      PRINT WORD OF "TREE" AND "TOP"
*PRINT WORD OF "TREE" AND "TOP"
TREETOP

NOW LET'S SEE HOW GOOD A COMPUTER YOU WOULD BE. @

WHEN YOU SEE THE ARROW, TYPE -- PRACTICE.
←PRACTICE

PRETEND YOU ARE THE COMPUTER. I WILL TYPE AN INSTRUCTION. YOU TYPE IN
 THE ANSWER THE COMPUTER WOULD GIVE. @

PRINT WORD OF "LA" AND "LA"
*LALA
GREAT! LET'S GO ON TO THE NEXT ONE. @

PRINT WORD OF "123" AND "4"
*1234
BRAVO! ON TO THE NEXT ONE! @

PRINT WORD OF "BE" AND "ARD"
*BEARD
GOOD FOR YOU. HERE'S THE FOURTH ONE. @

PRINT WORD OF "GOOD-" AND "BYE"
*GOOD-BYE
RIGHT YOU ARE. ONE MORE TO GO. @

PRINT WORD OF "E" AND WORD OF "N" AND "D"
*END
RIGHT YOU ARE, AND THAT IS THE END OF THIS LESSON. @

SO LONG FOR NOW.
←
```

```
←LESSON3

ANOTHER BUILT-IN LOGO PROCEDURE IS:   SENTENCE. @

TYPE THESE WORDS --     PRINT SENTENCE OF "GIRLS" AND "LAUGH"
*PRINT SENTENCE OF "GIRLS" AND "LAUGH"
GIRLS LAUGH

AS YOU SEE, SENTENCE PUTS TWO LOGO THINGS TOGETHER WITH A SPACE
BETWEEN. @

WHAT DO YOU THINK THE COMPUTER WOULD DO WITH THE NEXT INSTRUCTION? TYPE
IN JUST THE ANSWER THE COMPUTER WOULD GIVE. @

PRINT SENTENCE OF "BOYS" AND "PLAY"
*BOYS PLAY
GOOD FOR YOU.
HERE IS ANOTHER ONE. WHAT WOULD YOU REPLY?
PRINT SENTENCE OF "I LIKE" AND "ICE CREAM"
*I LIKE ICE CREAM
RIGHT YOU ARE.
THE PROCEDURE -- SENTENCE -- WILL PUT TOGETHER ONLY TWO LOGO THINGS AT
A TIME. @

LOOK AT WHAT YOU MUST DO TO HAVE THE COMPUTER PUT TOGETHER THREE
THINGS. @

TYPE THIS:     PRINT SENTENCE OF "I" AND SENTENCE OF "LIKE" AND "CANDY"
 * PRINT SENTENCE OF "I" AND SENTENCE OF "LIKE" AND "CANDY"
I LIKE CANDY

NOW TRY TO MAKE SOME SENTENCES OF YOUR OWN, WHEN YOU SEE THE ARROW.
←
```

MARCH 6 - The lesson was TEACHPRINT. The first thing I learned was that even with delays written into the procedure the typeout is too fast for these youngsters. I need to rewrite these procedures using a stop of some kind so that the youngsters can read at their own pace and then use the return key when they are ready for more reading material.* Greg finished TEACHPRINT and went on to TEACHWORD.

---

*All the lesson materials shown here and in the next section have incorporated this change.

Jay worked out the print instruction after a few false starts. He had trouble remembering matching quotation marks.

Ruth Anne, Shawn, and Julie played three games of THIRTY-ONE. None of them seemed to realize he could win.

MARCH 7 - All youngsters worked on TEACHWORD. Most of them had no difficulty with the change in directions which allowed them to control the rate at which information was presented. Karen kept asking "What shall I do now?" but this seemed to be more for support than from a real need for help. When I told her my lips were sealed, she went right over to strike the carriage return key for more information.

Although the operation WORD presented the children with no problem, my instructions definitely did. The children had trouble reading the word 'procedure' and they certainly did not understand what I was trying to say. An oral explanation seemed to clarify the issue.

The second and third graders have difficulty executing the instruction: "TYPE THE FOLLOWING:".

MARCH 10 - Steven worked his way through TEACHPRINT and TEACHWORD. I left him alone for a few minutes. He tried to type in some commands in unanticipated places. I am not sure how he interpreted the directions -- evidently he saw the stop points as invitations to type.

When Mary Jaye arrived I gave her a problem in which she had to use WORD. She wrote out the instruction at the blackboard, and then proceeded to write two or three more. Finally, I asked her to think about how to get three letters or parts of words together as one word. She and Steven both pondered this problem for a few minutes.

Then we worked it out in two parts. First we talked about WORD OF "C" AND "A". There was no doubt in their minds that this would produce CA. Then they also knew that WORD OF "CA" AND "T" would produce CAT. Finally we substituted. WORD OF "C" AND "A" is another name for CA. If we put this in place of "CA", we get PRINT WORD OF WORD OF "C" AND "A" AND "T". It worked.

```
←PRINT WORD OF "C" AND "A"
CA

←PRINT WORD OF "CA" AND "T"
CAT

←PRINT WORD OF WORD OF "C" AND "A" AND "T"
CAT
←
```

The other youngsters worked on TEACHSEN, a lesson for teaching the LOGO operation SENTENCE.

MARCH 12 - Jay began TEACHFIRLAS. I had asked him to type PRINT FIRST OF "SHE SELLS SEA SHELLS." He typed it in perfectly (we thought) but got an error comment. I typed it in and got an error comment too. I listed the procedure but could find nothing wrong with it. We went back and reexamined our work and sure enough, both of us had forgotten the period.

```
TYPE THESE WORDS AND LOOK VERY CAREFULLY AT THE COMPUTER'S
ANSWER:  PRINT FIRST OF "SHE SELLS SEA SHELLS."
*PRINT FIRST OF "SHE SELLS SEA SHELLS"
TRY AGAIN PLEASE.  THAT DOESN'T SEEM TO BE RIGHT.
```

There is a problem with writing clear directions. The youngsters have difficulty deciding when they are to type in an instruction and when they are simply to type in the result obtained from performing the instruction.

The LOGO lessons are very much alike, and perhaps a little monotonous, certainly not very creative or original. Yet, they do accomplish their intended purpose. The children are learning to understand the elementary LOGO operations, they seem to be happy with this kind of instruction, and I have learned a great deal about clarity of presentation and about learning difficulties.

MARCH 14 - All the children understand and freely use the operations FIRST, LAST, BUTFIRST, and BUTLAST. However, WORD and SENTENCE seem less easy for them perhaps because of the need for two inputs. Also, the children confuse a word in LOGO with the LOGO combining operation WORD.

MARCH 24 - The children worked on decoding the message in the LESSON MESSAGE with success and apparent pleasure. Everyone except Karen guessed the final message at least three lines before the end. When I suggested that perhaps we should stop and go on to something else, they were insistent that they be allowed to finish.

MARCH 26 - Everyone worked on TEACHCALL.

There are some difficulties with naming: e.g., (1) It seems more natural for children to put the name first, then the thing. (2) Children expect to be able to request the name of a thing as well as the thing of a name.

MARCH 27 - Mary Jaye worked for almost an hour today. We worked on CALL, which is difficult. She still likes to look at long strings of digits. I suggested that she try to keep her inputs short. She agreed with me and then typed out 20 and 30 character strings!

```
←PRINT BUTLAST OF "ABCDEFGHIJKLMNOP"
ABCDEFGHIJKLMNO
←CALL
   THING:  "97865432149777777777777777777777777777777"
   NAME:  "KEY"
←PRINT /KEY/
97865432149777777777777777777777777777777
←
```

Then she played THIRTY-ONE. Mary Jaye apparently cannot have too much of this, even though she didn't win. All her efforts were trial and error. It became apparent to her that 24 was a key number. She would comment ... "He's going to win, I think - he has 24." Then, however, she would suggest that maybe if she tried a different input when he had 24 she could still win. She also tried a couple of illegal inputs when she could not win with legal ones, just to see what would happen.

Greg tackled CALL with considerable authority and confidence. The CALLPRACTICE exercises generally worked better than the TEACHCALL lesson material proper. The children seemed to learn the effect of CALL by comparing different inputs and outputs.

When they make up their own names, the children have trouble remembering to use quotation marks (as I did when I was learning).

The children know that P stands for PRINT. I find, however, that they still type the whole word rather than the abbreviation.

MARCH 31 - Mary Jaye typed out the procedure INTRODUCE for me (I told her what to write) and watched what happened when we ran INTRODUCE using "MARY JAYE SIMMS" for /NAME/. We then worked on another procedure which she named TALK. She wrote a few instructions.

```
←TO TALK
>20 PRINT "121212121"
>30 PRINT "2323453454444444444"
>40 PRINT "TALK"
>50 PRINT "SANTA CLAUS"
>END
TALK DEFINED
```

Then I suggested testing it.

```
←TALK
121212121
2323453454444444444
TALK
SANTA CLAUS
```

She was delighted with it and edited it to add many more instructions.

```
←EDIT TAK \\LK
>60 PRINT "TOP AND HOP"
>70 PRINT"LOGO"
>80 PRINT "ADAM 12"
>90 PRINT "GOST"
>100 PRINT "123434343434345555555566666670809010020030"
>200 PRINT "12121212121212212121212121"
>300 PRINT "34567890000000000000001000000000000002000000000030000"
>400 PRINT "GOOD-BY"
>END
TALK DEFINED
```

(Finally she tested it with great pride and joy.)

```
←TALK
121212121
2323453454444444444
TALK
SANTA CLAUS
TOP AND HOP
LOGO
ADAM 12
GOST
123434343434345555555566666670809010020030
12121212121212212121212121
34567890000000000000001000000000000002000000000030000
GOOD-BY
←
```

Then we tried some naming again. Her work with long strings gave her some difficulty. For the first time I think she saw some value in working with brief words or symbols.

Jay also worked with INTRODUCE. I gave him models and he produced some of his own things using my model. In fact, we seemed to make so much progress that I threw in a second variable /AGE/. He obviously enjoyed using his friends' and family's names in these procedures. Before he ran each procedure, he would tell me just what the procedure was going to print out.

```
←LIST INTRODUE\CE

TO INTRODUCE /NAME/
10 PRINT /NAME/
20 PRINT /DATE/        (/DATE/ and /TIME/ are special LOGO
30 PRINT /TIME/         names for the current date and time,
END                     respectively.)

←INTRODUCE "JAY"
JAY
3/31/1969
1:27 PM

←EDIT INTRODUCE
>40 PRINT SENTENCE OF /NAME/ AND "IS VERY NICE"
>50 PRINT SENTENCE OF /NAME/ AND "GOES TO EMERSON SCHOOL"
>END
INTRODUCE DEFINED

←LIST INTRODUCE

TO INTRODUCE /NAME/
10 PRINT /NAME/
20 PRINT /DATE/
30 PRINT /TIME/
40 PRINT SENTENCE OF /NAME/ AND "IS VERY NICE"
50 PRINT SENTENCE OF /NAME/ AND "GOES TO EMERSON SCHOOL"
END
```

```
←INTRODUCE "LISA"
LISA
3/31/1969
1:43 PM
LISA IS VERY NICE
LISA GOES TO EMERSON SCHOOL

←TO DESCRIBE /NAME/ AND /AGE/
>10 PRINT /NAME/
>20 PRINT SENTENCE OF /AGE/ AND "YEARS OLD"
>30 PRINT SENTENCE OF /NAME/ AND "LIVES AT 10 CIRCET AVE"
>END
DESCRIBE DEFINED

←DESCRIBE "LISA" AND "6"
LISA
6 YEARS OLD
LISA LIVES AT 10 CIRCET AVE
←
```

Julie was back today after a considerable absence. Shawn acted as teacher and taught Julie about CALL. She caught on quickly.

APRIL 2 - Today, using TEACH-THE-COMPUTER, was the first time that the children made their own procedures. Mary Jaye, Jay, and the group of Ruth, Julie, and Shawn did very well. Greg kept asking about each step as if he had never seen any directions. Perhaps I should have had him read the directions aloud. This seems to help. When he did finally write a procedure of his own, it was the only one that was not almost a carbon copy of SPELLCAT, the one I had written for demonstration.

Steven did almost two lessons, to make up for his absences. It is amazing that he remembered the exact names he had used and all the work he had done previously.

APRIL 3 - Greg remembered our initial work with procedures which kept typing down the page. He really wrote the procedure

KEEPSADSACKING by himself - though he looked to me for confirmation at every step.

```
←TO SADSACK
>10 PRINT "HI"
>20 PRINT "ZOOM"
>END
SADSACK DEFINED

←PRINT "HI ZOOM"     (I am not sure what he was thinking here)
HI ZOOM

←PRINT "SADSACK"
SADSACK

←SADSACK             (He finally worked this out)
HI
ZOOM

←TO KEEPSADSACKING
>10 SADSACK
>20 KEEPSADSACKING
>END
KEEPSADSACKING DEFINED

←KEEPSADSACKING
HI
ZOOM
HI
ZOOM
HI
ZOOM
HI
ZOOM
HI
ZOOM                 (This printout went on for 4 pages)
 ⋮
```

I suggested to Jay that we work out a different kind of procedure. I showed him GROW (the printout only, not the program), thinking he might try to write a procedure which would do this.

```
←GROW
Z
ZZ
ZZZ
ZZZZ
ZZZZZ
ZZZZZZ
ZZZZZZZ
ZZZZZZZZ
```

He ended up with a similar idea, but a significant variation. He wrote the procedure GROWSMALL,

```
←TO GROWSMALL
>10 PRINT "EASTER"
>20 PRINT "ASTER"
>30 PRINT "STER"
>40 PRINT "TER"
>50 PRINT "ER"
>60 PRINT "R"
>END
GROWSMALL DEFINED
```

and then tried it out.

```
←GROWSMALL
EASTER
ASTER
STER
TER
ER
R
```

APRIL 7 - Today was game day. The children could select one game of their own choosing. The popular choice was THIRTY-ONE. I also taught most of them NIM. By and large they play at random. No one has really looked to see how the computer wins each time.

Greg was annoyed that he could not win at NIM. He was perhaps the only one to try to study what the computer did. He finally did win a game by emulating the computer but he had several false starts before he got there.

APRIL 10 - Today all of the group worked at least for a while on some debugging of programs. Greg was eager to go back to the SADSACK program he had written. He is delighted with the spewing out of line after line of print.

Jay worked on debugging COUNT-BY-TWO.

```
←LIST COUNT-BY-TWO

TO COUNT-BY-TWO
10 PRINT "2"
20 PRINT "6"
30 PRINT "10"
END

←EDIT COUNT-BY-TWO
>15 PRINT "4"
>25 PRINT "8"
>END
COUNT-BY-TWO DEFINED

←COUNT-BY-TWO
2
4
6
8
10
```

APRIL 14 - I have been working with a disturbed third grader who was expelled from another school last year in the second grade. He knows and uses every four-letter word in the book. The first time the computer did not respond as he wished (during a game of tic-tac-toe), he typed in ---- (not reproduced here). The computer responded ---- IS NOT DEFINED. However, since that time he has become protective of the terminal. Another youngster, traveling through at some time when the room was vacant, left his imprint on the paper, a rather mild expletive. My student was indignant and proceeded to dispose of the paper quickly.

Steven and Mary Jaye both worked on LESSON TEN. Steven needed no explanations from me at all until the very end when he needed to talk about FIRST OF FIRST OF a sentence. It is a pleasure to watch him at work.

Mary Jaye ran into a few more problems than Steven did but she worked her way through them on her own, with great success. Both of them guessed the message but both wanted to finish the entire set because "it was fun." Mary Jaye completed debugging SPELLDOWN first. Together we analyzed Line 30 and then she wrote Line 20 in a flash with no help at all. I was astounded. I wonder now whether it was a wild guess or whether she really had a flash of insight.

APRIL 16 - Steven had earned his game day on Monday. He enjoyed HANGMAN but was annoyed when he was not successful. He started a NIM game while I was doing an errand. When I returned I found that he was working with an inordinately large number of X's. I suggested that he stop and restart with a more reasonable number since time was running out. He restarted with 7 X's and won -- which delighted him.

APRIL 28 - Conversation with Steven after he looked at the last part of LESSON TWELVE:

Steven: Doesn't it know how to DOUBLEFIRST?
Mrs. B.: No, it doesn't!
Steven (with great assurance!): That means I'll have to teach it how!

After he started to write DOUBLEFIRST, he decided that the title needed repairs. I had to show him how to edit this. Then I asked him: What are the parts you are going to put together? How do you put them together? He wrote the entire procedure by himself after these two rhetorical questions.

Karen is a puzzle! She seems to understand the syntax of some simple programs - but it was apparent that the concept of a procedure is still not clear. We went over the printout line by line to see where it came from. Then she added lines and told me with confidence where they should affect the printout.

Shawn wrote DOUBLEDOG, Julie wrote DOUBLEHA, and Ruth Anne wrote DOUBLERUTH-ANNE. They had no problems. They changed line numbers so their procedures would not be carbon copies of each other.

APRIL 30 - Greg wants and needs to be right and is annoyed if he makes a mistake. Despite this concern, however, he works very fast and often hits the return key before he has checked his line to be sure it is correct.

MAY 1 - Steven wrote several forms of TRIPLE today. He got a few complaints from the computer - the error comments were helpful to him. He wrote a procedure FIRSTLAST without help. He discovered on his own that he had failed to give the procedure an argument and corrected it himself.

```
←TRIPLE "BOY"
TRIPLE ISN'T DEFINED.
←TO TRIPLE "BOY"
YOU NEED / MARKS AROUND EACH ARGUMENT.
←TO TRIPLE /ANYWORD/
>16 PRINT WORD OF WORD OF /ANYWORD/ AND /ANYWORD/ AND /ANYWORD/
>END
TRIPLE DEFINED

←TRIPLE "DIET"
DIETDIETDIET
←TRIPLE "FOX"
FOXFOXFOX
```

```
←TO FIRSTLAST
>10 PRINT WORD OF FIRST OF /ANYWORD/ AND LAST OF /ANYWORD/
>EDIT TITLE
TITLE TO FIRSTLAST /ANYWORD/   (Changes TITLE line)
>END
FIRSTLAST DEFINED

←FIRSTLAST "SAM"
SM
```

MAY 5 - Jay was working through LESSON THIRTEEN on procedures which have one, two, and no inputs. I think he really was confused until he gave the computer the instruction, ADDON "TREE". When this turned out its own peculiar sentences, he suddenly seemed to catch on.

Today I gave Karen LESSON ELEVEN containing procedures to be debugged. She did pretty well. She started off hesitantly but was reasonably successful as she went through the lesson. This again was a lesson purely between Karen and the computer, and all of us were the better for it.

Greg struggled today - first to remember how to use DOUBLE and then to get a DOUBLEFIRST written. He needed a great deal of guidance and really could not have written this alone.

Shawn, Ruth Anne, and Julie looked at DOUBLE and tried it once. They then struggled with DOUBLEFIRST. They were so busy giving each other directions that I let them struggle. They ended up writing DOUBLE again, but only recognized this when they saw the output.

MAY 14 - We began some review work in preparation for our demonstration at the Spring Joint Computer Conference this Friday.

Karen was great today. She wrote DOUBLEDOG like a pro. I know that she worked from the model, DOUBLECAT, and copied this exactly, but this is progress for her.

Steven did some interesting work with SURPRISE-4 (LESSON FIFTEEN). He tried to list the procedure E but it had been made invisible to students. Then with considerable persistence he kept trying each new output as the next input. His theory was that perhaps the scrambling was so ordered that eventually the procedure E would return the letters of his name in the proper order. As he got to his last try, he said, "This is it, one way or another." He got the storybook ending - the letters of his last name appeared in proper order. Of the youngsters who tried this since, he was the only one to look for a pattern in the scrambling.

```
←LIST E

TO E /YOUR LAST NAME/

←E "EPSTEIN"
NIETEPS
←E "NIETEPS"
SPETNIE
←E "SPETNIE"
EINTSPE
←E "EINTSPE"
EPSTEIN
```

(The procedure E could not be listed. It had been rendered invisible.)

MAY 16 - Friday, the children demonstrated their work at live terminals for two hours at a special education meeting held as part of the A.F.I.P.S. Spring Joint Computer Conference in Boston. It was a long, exhausting day -- the kids were great and they loved every minute! I was concerned that they were going to be frightened and pressured by the crowds and the questions. They ate it up -- they turned out to be big showoffs!

MAY 19 - LESSON FIFTEEN seems to help the children to focus on the number of inputs a procedure needs. On the other hand, all the youngsters, except perhaps Steven, had difficulty remembering what to do if a procedure required no input.

MAY 21 - Steven was the first student today and had no problems writing TRIPLE on his own.

```
TO TPL /ANYNUMBER/
10 PRINT SUM OF SUM OF /ANYNUMBER/ AND /ANYNUMBER/ AND /ANYNUMBER/
END

←TPL "1000"
3000
←TPL "150"
450
```

Shawn, Ruth Anne, and Julie worked very well together today helping each other over the hurdles. They got carried away by working on the proper number of procedure inputs and forgot to specify the procedure they were using. They were able to help each other with this.

Mary Jaye still continues to use the long strings as input. I am not sure what appeals to her about these long numbers, but it is obvious that she does enjoy them.

```
←MYSTERY-6 "AUGUST" "13" "10"
MY BIRTHDAY IS AUGUST 13
I AM 10 YEARS OLD.
←MYSTERY-6 "HALLOWEEN" "1234343" "12345678910333330000000000"
MY BIRTHDAY IS HALLOWEEN 1234343
I AM 12345678910333330000000000 YEARS OLD.
←
```

Karen again appeared to be stumped by the request to write DOUBLE. However, when she was left alone, the next thing we heard was an "I did it! It worked!" She can do the job when there is no one around.

```
←TO DOUBLEKAREN
>10 PRINT WORD OF "KAREN" AND "KAREN"
>END
DOUBLEKAREN DEFINED
←DOUBLEKAREN
KARENKAREN
←
```

MAY 23 - Jay is trying now to work out his problems with the procedure TPL. TPL actually was supposed to triple a number by adding. Jay however wrote a word tripler, which was fine. He found out by himself that one of his bugs yesterday was his failure to specify an input. Today he put that in immediately.

```
←TPL "TREE"
TREETREETREE
←TPL "4"
444
←TPL "TEE"
TEETEETEE
←
```

May 28 - Steven began by reviewing conditionals. He worked out the first one himself. Then I suggested that he try - IS /GREEN/ /BLUE/. As he was typing it in, he said to me, "I know why you want me to try this one. You think I'll say no because they are different letters, but I know they are the same." (In this exercise they both name the empty word.)

Perhaps any of the children could make great progress in LOGO if time permitted. The periods seem too short. For the next two weeks, I would like to work for half-hour periods with four of the children to see what can be done and how quickly they can move. Even a half-hour is not much time, but it should help.

* * * * * * * *

No daily log was kept for the period May 29 - June 13. The group was reduced to four children - Ruth Anne, Mary Jaye, Jay, and Steven - during these last two weeks. They finished working through the remaining lessons and spent the rest of the time writing LOGO procedures. For the most part, these followed very closely on the pattern of prescribed models. Thus, given the procedure

```
TO SAYHI
10 PRINT "HI"
20 SAYHI
END
```

whose effect was the endless stream

```
HI
HI
HI
:
```

all of the students were already able to make a (virtually duplicate) procedure for printing some other message. Here is one of Ruth Anne's.

```
TO SAYRUTH-ANNE
10 PRINT "RUTH-ANNE"
20 SAYRUTH-ANNE
END
```

Its effect is to print

```
RUTH-ANNE
RUTH-ANNE
RUTH-ANNE
  :   :
```

By the end of the course, they all were able to write a single procedure for printing any specified message. For example, the following procedure of Jay's:

```
TO SAY /ANYWORD/
10 PRINT /ANYWORD/
20 SAY /ANYWORD/
END
```

On the last day, Jay used his procedure as follows,

```
SAY "GOODBYE MISS BLOOM"
```

which generated

```
GOODBYE MISS BLOOM
GOODBYE MISS BLOOM
GOODBYE MISS BLOOM
       .
       .
       .
```

Mary Jaye's last procedure, virtually the same as Jay's, produced this farewell message.

```
I WILL MISS YOU
I WILL MISS YOU
I WILL MISS YOU
      .
      .
      .
```

During these last days, simple recursive procedures such as SAY and WORKLESS were favorites with all the children. They would write them anew,

```
←TO WORKLESS /ANYWORD/
>10 PRINT /ANYWORD/
>29 CALL
   THING:  BUTFIRST OF /ANYWORD/
   NAME:  "NEWWORD
>30 WORKLESS /NEWWORD/
>END
WORKLESS DEFINED
```

and then run them repeatedly. These are relatively small printouts:

```
←WORKLESS "I AM A BOY"
I AM A BOY
AM A BOY
A BOY
BOY

←WORKLESS "!#$%&'()*=<>?+@[GS"
!#$%&'()*=<>?+@[GS
#$%&'()*=<>?+@[GS
$%&'()*=<>?+@[GS
%&'()*=<>?+@[GS
&'()*=<>?+@[GS
'()*=<>?+@[GS
()*=<>?+@[GS
)*=<>?+@[GS
*=<>?+@[GS
=<>?+@[GS
<>?+@[GS
>?+@[GS
?+@[GS
+@[GS
@[GS
[GS
GS
S
```

they all liked to make big ones.

```
←WORKLESS "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ1234567899"
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ1234567899
BCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ1234567899
CDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ1234567899
DEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ1234567899
     .                           .
     .                           .
     .                           .
ZABCDEFGHIJKLMNOPQRSTUVWXYZ1234567899
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567899
BCDEFGHIJKLMNOPQRSTUVWXYZ1234567899
CDEFGHIJKLMNOPQRSTUVWXYZ1234567899
     .
     .
     .
Z1234567899
1234567899
234567899
34567899
4567899
567899
67899
7899
899
99
9
←
```

Within the remaining time span, only one child - Steven - was able to use LOGO to go beyond the transliteration of prescribed models for simple procedures. On June 4, he started working on his own procedure, COUNTDOWN, which was to record the integers as they descend to zero (prior to blast-off). He began by writing

```
←TO COUNTDOWN /ANYNUMBER/
>10 PRINT /ANYNUMBER/
>20 CALL
     THING:  DIFF OF /ANYNUMBER/ AND /ANYNUMBER/
     NAME:  "NEWNUMBER"
>30 COUNTDOWN /NEWNUMBER/
>8 IS /ANYNUMBER/ /EMPTY/  (He suddenly realized he needed
>9 IF YES RETURN /EMPTY/   a test for stopping the countdown
>END                       and it had to precede Line 10)
COUNTDOWN DEFINED
```

Then he tried out his procedure.

```
←COUNTDOWN 5
5
0
0
0
.
.
.
```

Something was wrong. He saw the first bug, a wrong subtraction input in Line 20, and he fixed it.

```
←EDIT COUNTDOWN
>20 CALL
     THING:  DIFF OF /ANYNUMBER/ AND "1"
     NAME:  "NEWNUMBER"
>END
COUNTDOWN DEFINED
```

Then he tried again.

```
←COUNTDOWN "7"
7
6
5
4
3
2
1
0
-1
-2
-3
  :
  :
```

Still something wrong. It continued printing past zero into the negative numbers. So he fixed the second bug, by changing the stop rule in Line 8,

```
←EDIT COUNTDOWN
>8 IS /ANYNUMBER/ "-1"
>END
COUNTDOWN DEFINED
```

and then tried once more.

```
←COUNTDOWN "4"
4
3
2
1
0
```

And it worked!

His next assignment was to make a procedure for counting down by two's. Steven said, "Oh, I know how to do that." Then he wrote COUNTDOWN-2 using COUNTDOWN as his model. He changed the difference operation correctly (Line 20) but he did not change the stop rule.

```
TO COUNTDOWN-2 /ANYNUMBER/
8 IS /ANYNUMBER/ "-1"
9 IF YES RETURN /EMPTY/
10 PRINT /ANYNUMBER/
20 CALL
    THING:  DIFFERENCE OF /ANYNUMBER/ AND "2"
    NAME:  "NEWNUMBER"
30 COUNTDOWN-2 /NEWNUMBER/
END
COUNTDOWN-2 DEFINED
```

So, when he ran his COUNTDOWN-2,

```
←COUNTDOWN-2 "8"
8
6
4
2
0
-2
-4
:
```

it did not stop at 0.

He spotted his bug immediately. He started to fix it and then said, "but, I need to keep the 'one'. I might want to start with an odd number."

```
←EDIT COUNTDOWN-2
>6 IS /ANYNUMBER/ "2"
>7 IF YES RETURN /EMPTY/
>END
COUNTDOWN-2 DEFINED
```

(Note that he has added a new stop rule in Lines 6 and 7, but he has kept Lines 8 and 9.)

```
←COUNTDOWN-2 "8"
8
6
4
2
0
←COUNTDOWN-2 "7"
7
5
3
1
←
```

(So his program will stop for odd as well as even numbers.)

His next assignment was to count <u>up</u> from any given number and stop at 20. No faltering this time.

```
←TO COUNTUP /ANYNUMBER/
>8 IS /ANYNUMBER/ "21"
>9 IF YES RETURN /EMPTY/
>10 PRINT /ANYNUMBER/
>20 CALL
     THING:  SUM OF /ANYNUMBER/ AND "1"
     NAME:  "NEWNUMBER"
>30 COUNTUP /NEWNUMBER/
>END
COUNTUP DEFINED
```

His first program worked.

```
←COUNTUP "15"
15
16
17
18
19
20
←
```

Steven saved his countup and countdown programs in a LOGO file. Each day when he came to class, he got a copy of his programs from the file and ran each of them with new inputs. Then he worked on some new variations. For example, he wrote a procedure for counting down by threes, and a procedure for counting up to numbers larger than 20.

His last assignment, on June 13, was to write a procedure for counting down from any given number to any given lower number. He went right to it.

```
←TO COUNTDOWN /ANYNUMBER/ AND /LOWNUMBER/
>8 IS /ANYNUMBER/ /LOWNUMBER/
>9 IF YES RETURN /EMPTY/
>10 PRINT /ANYNUMBER/
>20 CALL
     THING:  DIFFERENCE OF /ANYNUMBER/ AND 1
     NAME:  "NEWNUMBER"
>30 COUNTDOWN /NEWNUMBER/ AND /LOWNUMBER/
END
```

It looked right.

```
←COUNTDOWN "12" "7"
12
11
10
9
8
←
```

But, it stopped a little too soon!

The stop rule had to be changed, but that was easy.

```
←EDIT COUNTDOWN
>8 IS /ANYNUMBER/ DIFF OF /LOWNUMBER/ AND 1
>END
COUNTDOWN DEFINED
```

Now it would work,

```
←COUNTDOWN "12" "7"
12
11
10
9
8
7
←
```

- even with negative numbers!

```
←COUNTDOWN "6" "-3"
6
5
4
3
2
1
-1
-2
-3
←
```

At this point, his work ended. That's all there was time for him to do.

3.3 The Lesson Materials

The children's time in the course was mainly spent on working through a series of about twenty lessons programmed in LOGO. This section contains printouts of children's interactions with each of these lesson programs. These typescripts have been included in the report not merely for historical recording and documentation of the work but because they give very specific insights into the problems of teaching, and the experience of learning, a formal language through a somewhat open-ended mechanical presentation. The proscriptive parts of the material incorporate a great number of problems "to make" as well as questions "to answer". Also, the later lessons require the children to write procedures on their own. Thus these lesson interactions reveal something about the children, as well as the teacher.

The content treated in the lessons is summarized as follows.

LESSON ONE. The command PRINT used with a literal

LESSON TWO. The operation WORD used with the PRINT command.

LESSON THREE. The operation SENTENCE used with the PRINT command.

LESSON FOUR. The two operations FIRST and LAST as applied to words and sentences.

LESSON FIVE. The two operations BUTFIRST and BUTLAST as applied to words and sentences.

LESSON MESSAGE. A secret message is decoded by exercising the operations introduced in the previous lessons.

LESSON SIX. The CALL command: LOGO things and names. Practice in naming and the use of names.

LESSON SEVEN (TEACH-THE-COMPUTER). The command TO. Writing and performing some simple procedures.

LESSON EIGHT (EXPLAIN). Debugging six simple procedures.

LESSON NINE. The operations COUNT, SUM, and DIFFERENCE reviewed. (These operations were first taught orally during some earlier periods at the terminal.)

LESSON TEN. A review of FIRST, LAST, BUTFIRST, BUTLAST, and WORD. The answers are used to spell out another secret message.

LESSON ELEVEN. Debugging practice with some pattern drawing procedures.

LESSON TWELVE. Going from a procedure for doubling a specific word to a general doubling procedure which can take any word as its input.

REVIEW-1. Practice in simulating the operation of a procedure.

LESSON THIRTEEN (USEINPUT). Extending the number of inputs of procedures.

LESSON FOURTEEN. Writing a procedure as a command, and as an operation.

LESSON FIFTEEN. Eleven procedures illustrating strange and funny inputs.

LESSON SIXTEEN. Practice in writing procedures. The effects of several procedures are described in succession and sample printed results are shown for each. The student attempts to write each of the procedures.

LESSON SEVENTEEN. Practice with the identity operation IS.

LESSON EIGHTEEN. Writing a general recursive procedure.

The interactions are easy to read through. We have underscored all of the children's typing to distinguish it from the computer's, and have incorporated occasional expository comments (in parentheses) particularly in the later lessons.

These lessons were written in LOGO. The LOGO programs for Lesson ONE and Lesson EIGHTEEN, the first and last in the series, are reproduced and discussed at the end of this section.

```
←LESSON1

IN THIS LESSON THERE ARE SOME LINES TO READ. THE COMPUTER TYPES VERY
FAST. IT WILL STOP EVERY SO OFTEN SO THAT YOU WILL HAVE TIME TO READ.
WHEN YOU WANT IT TO GO ON, PRESS THE RETURN KEY. @

THIS MARK @ MEANS THAT THE COMPUTER HAS MORE TO TELL YOU. PRESS THE
RETURN KEY WHEN YOU ARE READY TO READ IT. @

THE COMPUTER UNDERSTANDS SOME SPECIAL COMMANDS. THE FIRST ONE WE WILL
TALK ABOUT IS -- PRINT. @

LET'S TELL THE COMPUTER TO PRINT A WORD. I WILL TYPE THE COMMAND. YOU
PRESS THE RETURN KEY WHEN I AM FINISHED.

PRINT "CABBAGE"
CABBAGE

NOW LET'S TELL THE COMPUTER TO PRINT A SENTENCE. YOU PRESS THE RETURN
KEY THIS TIME TOO.

PRINT "DO YOU LIKE THE BEATLES?"
DO YOU LIKE THE BEATLES?

NOTICE THAT THERE WERE QUOTATION (" ") MARKS AROUND THE EXACT WORDS THE
COMPUTER WAS ASKED TO PRINT. @

NOW TRY TO MAKE THE COMPUTER PRINT SOME OF YOUR OWN THINGS. @

DON'T WORRY ABOUT MAKING MISTAKES! WE ALL DO IT! THE COMPUTER WILL TRY
TO HELP YOU BY EXPLAINING WHAT WENT WRONG. @

HAVE FUN!
←
```

```
←LESSON2

REMEMBER -- THE MARK @ MEANS THAT I AM WAITING FOR YOU TO PRESS THE
RETURN KEY. @

AFTER YOU LEARN TO USE LOGO, YOU WILL BE ABLE TO TEACH THE COMPUTER TO
DO NEW THINGS. @

YOU WILL TEACH THE COMPUTER TO DO SOMETHING NEW BY WRITING A PROCEDURE.
@

A PROCEDURE IS LIKE A RECIPE. IT HAS ALL THE INGREDIENTS AND DIRECTIONS
IN IT TO MAKE SOMETHING. @

A FEW PROCEDURES ARE ALREADY BUILT INTO LOGO. WE WILL TALK ABOUT THESE
BEFORE WE TALK ABOUT PROCEDURES THAT YOU WILL WRITE. @

THE PROCEDURE WE WILL LEARN ABOUT IN THIS LESSON IS CALLED -- WORD.
LET'S SEE WHAT IT DOES. @

TYPE THESE WORDS --      PRINT WORD OF "UP" AND "DOWN"
*PRINT WORD OF "UP" AND "DOWN"
UPDOWN

HERE IS ANOTHER EXAMPLE OF WHAT -- WORD -- DOES.

TYPE THIS:      PRINT WORD OF "TREE" AND "TOP"
*PRINT WORD OF "TREE" AND "TOP"
TREETOP

NOW LET'S SEE HOW GOOD A COMPUTER YOU WOULD BE. @

WHEN YOU SEE THE ARROW, TYPE -- PRACTICE.
←
←PRACTICE

PRETEND YOU ARE THE COMPUTER. I WILL TYPE AN INSTRUCTION. YOU TYPE IN
THE ANSWER THE COMPUTER WOULD GIVE. @

PRINT WORD OF "LA" AND "LA"
*LALA
GREAT! LET'S GO ON TO THE NEXT ONE. @

PRINT WORD OF "123" AND "4"
*1234
BRAVO! ON TO THE NEXT ONE! @
```

```
PRINT WORD OF "BE" AND "ARD"
*BEARD
GOOD FOR YOU. HERE'S THE FOURTH ONE. @

PRINT WORD OF "GOOD-" AND "BYE"
*GOOD-BYE
RIGHT YOU ARE. ONE MORE TO GO. @

PRINT WORD OF "E" AND WORD OF "N" AND "D"
*END
RIGHT YOU ARE, AND THAT IS THE END OF THIS LESSON. @

SO LONG FOR NOW.
←



←LESSON3

ANOTHER BUILT-IN LOGO PROCEDURE IS:   SENTENCE. @

TYPE THESE WORDS --     PRINT SENTENCE OF "GIRLS" AND "LAUGH"
*PRINT SENTENCE OF "GIRLS" AND "LAUGH"
GIRLS LAUGH

AS YOU SEE, SENTENCE PUTS TWO LOGO THINGS TOGETHER WITH A SPACE
BETWEEN. @

WHAT DO YOU THINK THE COMPUTER WOULD DO WITH THE NEXT INSTRUCTION? TYPE
 IN JUST THE ANSWER THE COMPUTER WOULD GIVE. @

PRINT SENTENCE OF "BOYS" AND "PLAY"
*BOYS PLAY
GOOD FOR YOU.
HERE IS ANOTHER ONE. WHAT WOULD YOU REPLY?
PRINT SENTENCE OF "I LIKE" AND "ICE CREAM"
*I LIKE ICE CREAM
RIGHT YOU ARE.
THE PROCEDURE -- SENTENCE -- WILL PUT TOGETHER ONLY TWO LOGO THINGS AT
A TIME. @

LOOK AT WHAT YOU MUST DO TO HAVE THE COMPUTER PUT TOGETHER THREE
THINGS. @

TYPE THIS:     PRINT SENTENCE OF "I" AND SENTENCE OF "LIKE" AND "CANDY"
 *PRINT SENTENCE OF "I" AND SENTENCE OF "LIKE" AND "CANDY"
I LIKE CANDY

NOW TRY TO MAKE SOME SENTENCES OF YOUR OWN, WHEN YOU SEE THE ARROW.
←
```

```
←LESSON4

TYPE THIS:     PRINT FIRST OF "PURPLE"
*PRINT FIRST OF "PURPLE"
P

NOW TYPE THIS:     PRINT FIRST OF "TREE"
*PRINT FIRST OF "TREE"
T

TYPE THESE WORDS AND LOOK VERY CAREFULLY AT THE COMPUTER'S ANSWER:
PRINT FIRST OF "SHE SELLS SEA SHELLS"
*PRINT FIRST OF "SHE SELLS SEA SHELLS"
SHE

I'LL BET YOU EXPECTED TO SEE 'S' INSTEAD OF 'SHE'. @

THE PROCEDURE -- FIRST -- TELLS THE COMPUTER TO OUTPUT THE FIRST LETTER
IF THE INPUT IS A WORD.  IT TELLS THE COMPUTER TO OUTPUT THE FIRST WORD
IF THE INPUT IS A SENTENCE. @

NOW LET'S LOOK AT ANOTHER PROCEDURE CALLED -- LAST. @

TYPE THIS:     PRINT LAST OF "FOO"
*PRINT LAST OF "FOO"
O

HERE IS ANOTHER ONE TO TYPE:     PRINT LAST OF "KETCHUP"
*PRINT LAST OF "KETCHUP"
P

HERE IS THE LAST ONE:     PRINT LAST OF "UP, UP, AND AWAY"
*PRINT LAST OF "UP, UP, AND AWAY"
AWAY

NOW TELL THE COMPUTER TO DO SOME THINGS OF YOUR OWN WITH FIRST AND
LAST.
←
```

```
←LESSONS

TYPE THESE WORDS --     PRINT BUTFIRST OF "TICKLE"
*PRINT BUTFIRST OF "TICKLE"
ICKLE

NOW TRY THIS:     PRINT BUTFIRST OF "SEESAW"
*PRINT BUTFIRST OF "SEESAW"
EESAW

BUTFIRST IS A PROCEDURE WHICH TELLS THE COMPUTER TO OUTPUT EVERY LETTER
BUT THE FIRST LETTER OF A WORD. @

NOW LET'S SEE WHAT HAPPENS IF WE ASK FOR BUTFIRST OF A SENTENCE. @

TYPE THIS:     PRINT BUTFIRST OF "SNOOPY AND CHARLIE BROWN"
*PRINT BUTFIRST OF "SNOOPY AND CHARLIE BROWN"
AND CHARLIE BROWN

BUTFIRST TELLS THE COMPUTER TO OUTPUT EVERY WORD EXCEPT THE FIRST WORD
OF A SENTENCE. @

HERE IS STILL ANOTHER BUILT-IN LOGO PROCEDURE -- BUTLAST.@

TYPE THIS:     PRINT BUTLAST OF "JEEPERS"
*PRINT BUTLAST OF "JEEPERS"
JEEPER

NOW TRY ANOTHER --    PRINT BUTLAST OF "LUCY"
*PRINT BUTLAST OF "LUCY"
LUC

NOW TRY BUTLAST WITH A SENTENCE:     PRINT BUTLAST OF "FEE FI FO FUM"
*PRINT BUTLAST OF "FEE FI FO FUM"
FEE FI FO

NOW YOU ARE ON YOUR OWN FOR A WHILE. TRY LOTS OF YOUR OWN INSTRUCTIONS
WITH WORD, SENTENCE, FIRST, LAST, BUTFIRST, AND BUTLAST.
←
```

LESSON MESSAGE

```
I HAVE A MYSTERY MESSAGE FOR YOU. I WILL GIVE YOU THE MESSAGE WITH
DASHES WHICH STAND FOR THE LETTERS IN EACH WORD. PRESS THE RETURN KEY
W HEN YOU ARE READY FOR IT.

--- --- ----- - ---- --- -------- ----. -- ---- ---- -- ---.

YOU CAN DECODE THE MESSAGE BY PRETENDING YOU ARE THE COMPUTER.
WHEN YOU SEE THE *, TYPE IN THE ANSWER YOU THINK A COMPUTER WOULD GIVE.
IF YOU ARE READY FOR THE CHALLENGE, PRESS THE RETURN KEY.
PRINT LAST OF "DONKEY"
*Y

Y-- --- ----- - ---- --- -------- ----. -- ---- ---- -- ---.

PRINT WORD OF "E" AND "A"
*EA

Y-- --- ----- - ---- --- -EA----- ----. -- ---- ---- -- ---.

PRINT BUTFIRST OF "RING"
*ING

Y-- --- --ING - ---- --- -EA--ING ----. -- ---- ---- -- ---.

      :    :

PRINT FIRST OF "JAMBOREE"
*J

YOU ARE -OING - GOO- JO- -EAR-ING -OGO. -E -O-E THIS IS FUN.

PRINT FIRST OF "WIGGLE"
*W

YOU ARE -OING - GOO- JO- -EAR-ING -OGO. WE -O-E THIS IS FUN.

PRINT LAST OF "BAWL"
*L

YOU ARE -OING - GOO- JO- LEAR-ING LOGO. WE -O-E THIS IS FUN.

PRINT BUTLAST OF "AS"
*A

YOU ARE -OING A GOO- JO- LEAR-ING LOGO. WE -O-E THIS IS FUN.

      :    :
```

```
←TEACHCALL  (LESSON SIX)

UNTIL NOW YOU HAVE USED QUOTATION MARKS (" ") AROUND EACH INPUT TO TELL
T HE COMPUTER THE WORD OR SENTENCE TO USE. @
HERE IS AN EXAMPLE -- YOU TYPE:    PRINT SENTENCE OF "DATE" AND "TIME"
* PRINT SENTENCE OF "DATE" AND "TIME"
DATE TIME

SOMETIMES WE DON'T USE QUOTATION MARKS. WE USE SLASHES INSTEAD. THE
SLASH MARKS HAVE A DIFFERENT MEANING. @

YOU TYPE THIS:      PRINT SENTENCE OF /DATE/ AND /TIME/
*PRINT SENTENCE OF /DATE/ AND /TIME/
3/26/1969 12:43 PM


AS YOU CAN SEE, THERE IS A DIFFERENCE BETWEEN "DATE" AND /DATE/ AND
BETWEEN "TIME" AND /TIME/. @

THE SLASH MARKS TELL THE COMPUTER THAT DATE IS A NAME FOR SOMETHING
ELSE, AND THAT TIME IS A NAME FOR SOMETHING ELSE. THE COMPUTER MUST
LOOK TO SEE IF IT HAS BEEN TAUGHT WHAT DATE STANDS FOR AND WHAT TIME
STANDS FOR. @

THE COMPUTER SHOWED THAT IT HAD ALREADY BEEN TAUGHT THAT DATE IS THE
NAME OF 3/26/1969 AND THAT TIME IS A NAME FOR 12:43 PM. BUT YOU CAN
TEACH IT NEW NAMES FOR OTHER THINGS. @

LET US TEACH THE COMPUTER THAT THE WORD "GRRR" IS TO BE A NAME FOR THE
W ORD "GROWL". @

FIRST WE TYPE IN THE WORD -- CALL. THEN WE PRESS THE RETURN KEY. @

NOW THE COMPUTER WILL TYPE -- THING:  THEN WE TYPE THE WORD TO BE
NAMED.  IN THIS CASE WE ARE NAMING "GROWL". THEN WE PRESS THE RETURN
KEY AGAIN. @

THE COMPUTER WILL THEN TYPE -- NAME: THEN WE TYPE IN THE NAME WE ARE
GOING TO USE. IN THIS CASE IT IS "GRRR". WHEN WE HAVE FINISHED THE
NAMING, WE HIT RETURN AGAIN. @

LET ME DO THIS FOR YOU SO YOU CAN SEE HOW IT WORKS.@

CALL
  THING: "GROWL"
  NAME: "GRRR"
```

```
NOW YOU TYPE THIS:      PRINT /GRRR/
*PRINT /GRRR/
GROWL

DID YOU NOTICE THAT WE PUT QUOTATION MARKS AROUND THE THING WE NAMED
AND AROUND THE NAME WE USED? @


HERE IS ANOTHER ONE. BUT BEFORE WE DO THIS, YOU TRY A TEST. YOU TYPE:
   PRINT /SNOOPY/
*PRINT /SNOOPY/


DID YOU SEE THAT THE COMPUTER RETURNED EMPTY. IT DOESN'T KNOW WHAT
SNOOPY IS A NAME FOR BECAUSE WE HAVEN'T TAUGHT IT YET.

NOW LET'S MAKE SNOOPY THE NAME OF SOMETHING. @

CALL
  THING: "DOG"
  NAME: "SNOOPY"

NOW YOU TYPE:      PRINT /SNOOPY/
*PRINT /SNOOPY/
DOG

NOW TRY TO NAME A WORD OR SENTENCE OF YOUR OWN.
REMEMBER, WHEN YOU SEE *, TYPE CALL AND PRESS THE RETURN KEY.

*CALL
  THING:  "LADY"
  NAME: "MRS.BLOOM"
NOW TEST BY ASKING THE COMPUTER TO PRINT THE NAME YOU HAVE GIVEN IT,
WITH SLASH MARKS AROUND IT.
*PRINT   /MRS.BLOOM/
LADY

NOW TRY SOME OF THESE ON YOUR OWN.
←CALL
  THING:  "ONE"
  NAME: "ONE HUNDRED"
←PRINT "ONE HUNDRED"
ONE HUNDRED
←PRINT /ONE HUNDRED/
ONE
←
```

```
←PRACTCALL

I HAVE JUST TAUGHT THE COMPUTER SOME NEW NAMES. HERE THEY ARE:


CALL                            CALL
 THING: "GREEN"                  THING: "LEG"
 NAME: "STRING BEANS"            NAME: "HOT DOG"


CALL                            CALL
 THING: "BUMP"                   THING: "TWO"
 NAME: "BASEBALL"                NAME: "THREE"


CALL
 THING: "SHEEP"
 NAME: "SUGAR"


I AM GOING TO WRITE OUT SOME SENTENCES WITH SOME MISSING WORDS. @
YOU TYPE IN THE WORD WHICH SHOULD BE PUT IN THE BLANK SPACE SO THAT THE
S ENTENCE WILL MAKE SENSE. @
YOU MUST CHOOSE YOUR WORD FROM THE LIST ABOVE. @


1. I HATE VEGETABLES BUT MY MOTHER MAKES ME EAT "--------" ANYWAY.
*STRING BEANS
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
I HATE VEGETABLES BUT MY MOTHER MAKES ME EAT STRING BEANS ANYWAY.

2. HE FELL AND GOT A BIG /--------/ ON HIS HEAD.
*BUMP
HERE IS THE SENTENCE YOU JUST MADE:
HE FELL AND GOT A BIG ON HIS HEAD.
DOESN'T THIS SOUND FUNNY? PLEASE TRY AGAIN.
*BS\ASEBALL
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
HE FELL AND GOT A BIG BUMP ON HIS HEAD.

3. THE COLOR OF THE GRASS IS /--------/
*STRING BEANS
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
THE COLOR OF THE GRASS IS GREEN
```

```
4. I ATE A "--------" WITH MUSTARD ON IT.
*HOT DOG
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
I ATE A HOT DOG WITH MUSTARD ON IT.

5. ONE AND ONE ARE /--------/
*THREE
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
ONE AND ONE ARE TWO

6. HE SLIPPED ON THE ICE AND BROKE HIS /--------/.
*HOT DOG
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
HE SLIPPED ON THE ICE AND BROKE HIS LEG

7. THAT CHAIR HAS A WOBBLY "--------"
*LEG
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
THAT CHAIR HAS A WOBBLY LEG


8. SOME PEOPLE LIKE "--------" AND CREAM IN THEIR COFFEE.
*SUGAR
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
SOME PEOPLE LIKE SUGAR AND CREAM IN THEIR COFFEE.

9. LITTLE BO PEEP HAS LOST HER /--------/
*SUGAR
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
LITTLE BO PEEP HAS LOST HER SHEEP

10. THE RED SOX START PLAYING "--------" IN A FEW WEEKS.
*BASEBALL
YOU ARE ABSOLUTELY RIGHT. HERE IS YOUR SENTENCE:
THE RED SOX START PLAYING BASEBALL IN A FEW WEEKS.

THAT'S THE END OF MY LIST OF SENTENCES. SEE YOU LATER.
←
```

```
←TEACH-THE-COMPUTER   (LESSON SEVEN)

THE COMPUTER ALREADY KNOWS ABOUT SOME PROCEDURES LIKE WORD, SENTENCE,
FIRST, BUTLAST AND SOME OTHERS WE HAVE LEARNED. @

NOW LET'S TEACH IT SOMETHING NEW, THAT IT DOESN'T KNOW ALREADY. @

FIRST WE HAVE TO TELL IT THAT WE ARE GOING TO TEACH IT SOMETHING NEW. I
WILL SHOW YOU HOW TO DO THIS. @

FIRST I WILL TYPE THE WORD - TO.

TO

THEN I WILL GIVE THE NEW PROCEDURE A NAME. THE NAME OF MY NEW PROCEDURE
IS SPELLCAT. @

TO SPELLCAT

NOW I AM GOING TO TELL IT WHAT SPELLCAT IS SUPPOSED TO DO. I WANT
SPELLCAT TO DO THREE THINGS IN ORDER.@

I WILL NUMBER EACH STEP SO THE COMPUTER WILL KNOW THE RIGHT ORDER TO DO
THESE THINGS. @

TO SPELLCAT
10 PRINT "C"
20 PRINT "A"
30 PRINT "T"

NOW I HAVE TOLD THE COMPUTER THE THREE THINGS WHICH SPELLCAT SHOULD DO.
@

NOW I WILL TYPE END, TO LET THE COMPUTER KNOW I HAVE FINISHED TELLING
IT HOW TO SPELLCAT. @

TO SPELLCAT
10 PRINT "C"
20 PRINT "A"
30 PRINT "T"
END
```

```
THE COMPUTER WILL NOW TYPE BACK -- SPELLCAT DEFINED. IT IS TELLING ME
THAT IT NOW UNDERSTANDS SPELLCAT. @

HERE IS THE WAY IT WILL LOOK.

TO SPELLCAT
10 PRINT "C"
20 PRINT "A"
30 PRINT "T"
END
SPELLCAT DEFINED


NOW LET'S TRY THIS PROCEDURE TO SEE WHAT IT WILL DO. YOU TYPE -
SPELLCAT.
*SPELLCAT
C
A
T

NOW TRY WRITING YOUR OWN THING. HERE IS A SUGGESTION IF YOU NEED ONE. @

YOU MIGHT WRITE A PROCEDURE WHICH COUNTS LIKE THIS:

ONE
TWO
THREE
FOUR
FIVE

GOOD LUCK. HOPE YOU TEACH ME LOTS OF NEW FUN THINGS.
←TO NUMBER
>10 PRINT "O"
>20 PRINT "N"
>30 PRIMT\\NT "E"
>END
NUMBER DEFINED
←NUB\MBER
O
N
E
```

```
←EXPLAIN    (LESSON EIGHT)

HERE ARE SOME PROCEDURES THAT DON'T DO ALL THE THINGS THEY WERE
SUPPOSED TO DO.

THERE ARE SIX PROCEDURES. HERE ARE THEIR TITLES:

JOKE
HOPSKIP
COUNT-TO-FIVE
SPELLDOWN
REVCAT
REVDOG @

YOU ARE TO DO FIVE THINGS WITH EACH PROCEDURE: @

1. TRY OUT THE PROCEDURE BY TYPING ITS NAME. THIS WILL TELL YOU WHAT
THE PROCEDURE DOES NOW. @

2. TYPE THE PROCEDURE NAME AGAIN, BUT THIS TIME PUT A 1 AT THE END OF
THE NAME. EXAMPLE: JOKE1 - THIS WILL SHOW YOU WHAT THE PROCEDURE WAS
SUPPOSED TO DO. @

3. LIST THE PROCEDURE BY TYPING LIST ...... EXAMPLE: LIST REVDOG. THIS
W ILL LET YOU LOOK AT THE STEPS IN THE PROCEDURE SO YOU CAN SEE WHAT IS
M ISSING. @

4. EDIT THE PROCEDURE. YOU ARE TO CORRECT THE PROCEDURE TO MAKE IT
RIGHT. @

5. TEST THE PROCEDURE YOU JUST WROTE TO SEE IF IT DOES WHAT IT WAS
SUPPOSED TO DO. @

 JOKE
QUESTION: WHAT DID THE BIG CHIMNEY SAY TO THE LITTLE CHIMNEY?
←JOKE1
QUESTION: WHAT DID THE BIG CHIMNEY SAY TO THE LITTLE CHIMNEY?
ANSWER: YOU'RE TOO YOUNG TO SMOKE.
←LIST JOKE

TO JOKE
10 PRINT "QUESTION: WHAT DID THE BIG CHIMNEY SAY TO THE LITTLE CHIMNEY?"
END

←EDIT JOKE
>11 PRINT "ANSER:YOU'RE TO YOUNG TO SMOKE."
>END
JOKE DEFINED
←JOKE
QUESTION: WHAT DID THE BIG CHIMNEY SAY TO THE LITTLE CHIMNEY?
ANSER:YOU'RE TO YOUNG TO SMOKE.
←
```

```
 HOPSKIP
HOP

HOP
HOP
←HOPSKIP1
HOP

HOP

HOP
←

 LIST HOPSKIP

TO HOPSKIP
10 PRINT "HOP"
20 PRINT ""
30 PRINT "HOP"
50 PRINT "HOP"
END

←EDIT HOPSKIP
>40 PRINT ""
>END
HOPSKIP DEFINED
←HOPSKIP
HOP

HOP

HOP
←COUNT-TO-FIVE
ONE
THREE
FIVE
←COUNT-TO-FIVE1
ONE
TWO
THREE
FOUR
FIVE
```

```
←LIST COUNT-TO-FIVE

TO COUNT-TO-FIVE
10 PRINT "ONE"
30 PRINT "THREE"
50 PRINT "FIVE"
END

←EDIT COUNT-TO-FIVE
>20 PRINT "TWO"
>40 PRINT "FOUR"
>END
COUNT-TO-FIVE DEFINED
←COUNT-TO-FIVE
ONE
TWO
THREE
FOUR
FIVE
←

          :   :

 REVCAT
T
A
←REVCAT1
T
A
C
←

 LIST REVCAT

TO REVCAT
10 PRINT LAST OF "CAT"
20 PRINT LAST OF BUTLAST OF "CAT"
END

←EDIT REVCAT
>30 PRINT FIRSY\T OF "CAT"
>END
REVCAT DEFINED
←REVCAT
T
A
C
←
```

```
←USENUMBERS   (LESSON NINE)

THIS LESSON IS TO HELP YOU REVIEW SOME PROCEDURES CALLED COUNT, SUM,
AND DIFFERENCE. @

TO MAKE THE TYPING EASIER, WE WILL USE THE ABBREVIATION FOR DIFFERENCE:
DIFF. @

PRETEND YOU ARE THE COMPUTER. GIVE THE ANSWER YOU THINK THE COMPUTER
WOULD GIVE, IN EACH CASE. @

PRINT COUNT OF "BOP"
*3
GOOD FOR YOU. HERE IS THE NEXT ONE.

PRINT COUNT OF "ELEPHANT"
*7
NOT WHAT THE COMPUTER WOULD HAVE DONE. GIVE IT ANOTHER WHIRL.
*8
GOOD FOR YOU. HERE IS THE NEXT ONE.

PRINT COUNT OF "INDIANAPOLIS"
*12
GOOD FOR YOU. HERE IS THE NEXT ONE.

PRINT COUNT OF "THIS MONTH IS APRIL."
*16
LOOK AGAIN, AND TRY AGAIN PLEASE
*4
GOOD FOR YOU. HERE IS THE NEXT ONE.

PRINT COUNT OF "APRIL"
*5
GOOD FOR YOU. HERE IS THE NEXT ONE.

PRINT SUM OF "8" AND "7"
*15
GOOD FOR YOU. HERE IS THE NEXT ONE.
PRINT SUM OF "20" AND "100"
*300
I DON'T THINK THE COMPUTER AGREES WITH YOU. PLEASE TRY AGAIN.
*120
GOOD FOR YOU. HERE IS THE NEXT ONE.

PRINT DIFF OF "12" AND "8"
*4
GOOD FOR YOU. HERE IS THE NEXT ONE.
     :    :
```

```
←DECODE   (LESSON TEN)

HERE IS ANOTHER MESSAGE FOR YOU TO DECODE. @

-------- -- ---- --- --- ---. --- --- ---- -------, ----------, ---, --
- ------------.

PRETEND YOU ARE THE COMPUTER AND GIVE THE ANSWER YOU THINK THE COMPUTER
W OULD GIVE.@
          :   :
          :   :
PRINT BUTFIRST OF BUTLAST OF "HORN"
*OR
---OR-OW -- ---- --- -OR ---. --- --- ---- -------, ----------, ---, OR
- ----------OW.
PRINT BUTFIRST OF BUTLAST OF "FOUL"
*OU
---OR-OW -- ---- --- -OR -OU. -OU --- ---- -------, ----------, ---, OR
-OU--------OW.
PRINT LAST OF "FLY"
*Y
---OR-OW -- ---- --Y -OR YOU. YOU --- ---Y -------, -----Y----. ---, OR
- OU--------OW.
PRINT WORD OF FIRST OF "IT" AND LAST OF "BOSS"
*IS
---OR-OW IS ---- --Y -OR YOU. YOU --- ---Y -------, -----Y----, ---, OR
- OU--------OW.
PRINT FIRST OF BUTFIRST OF "AGREE"
*G
---OR-OW IS G--- --Y -OR YOU. YOU --- ---Y ---G---, -----Y----, OR
-OU--------OW.
PRINT BUTFIRST OF BUTFIRST OF "PLAN"
*AN
---OR-OW IS G--- --Y -OR YOU. YOU -AN ---Y -ANG-AN, -----Y----, --- OR
- OU--------OW.

          :   :
          :   :

PRINT BUTFIRST OF BUTLAST OF BUTLAST OF "SPLASH"
*PLA
TOMORROW IS GAME DAY FOR YOU. YOU CAN PLAY HANGMAN, THIRTY-ONE, NIM, OR
FOUR-IN---ROW.
PRINT FIRST OF LAST OF "THIS IS ALL."
*A
TOMORROW IS GAME DAY FOR YOU. YOU CAN PLAY HANGMAN, THIRTY-ONE, NIM, OR
F OUR-IN-A-ROW.
CONGRATULATIONS. YOU HAVE SOLVED THE MYSTERY. HAVE YOU DECIDED WHAT YOU
W ANT TO PLAY TOMORROW?
←
```

```
←PRACTICE   (LESSON ELEVEN)
I AM GOING TO SHOW YOU SOME PROCEDURES WHICH NEED TO BE FIXED. I HOPE
YOU CAN HELP ME. @

FIRST I WILL SHOW YOU WHAT THE PROCEDURE DOES.@

THEN I WILL LIST THE PROCEDURE. @

THEN I WILL ASK FOR HELP. YOU CAN HELP ME BY GIVING ME THE LINE NUMBER
WHICH WILL ANSWER THE QUESTION. @

HERE WE GO! IF YOU ARE PUZZLED, TRY A FEW AND YOU WILL SEE HOW THIS
WORKS. @

THE FIRST PROCEDURE IS CALLED SHRINK. I WILL SHOW YOU WHAT IT DOES.

SHRINK
XXXXXXX
XXXXXX
XXXX
XXXX
XXX
XX
X

HERE IS A LISTING OF THE PROCEDURE.

LIST SHRINK

TO SHRINK
10 PRINT "XXXXXXX"
20 PRINT "XXXXXX"
30 PRINT "XXXX"
40 PRINT "XXXX"
50 PRINT "XXX"
60 PRINT "XX"
70 PRINT "X"
END

QUESTION: WHICH LINE SHOULD BE CHANGED SO THAT PATTERN WILL LOOK RIGHT?
*30

THANK YOU. WILL YOU HELP ME ON THE NEXT ONE?
```

```
HERE IS ANOTHER PROCEDURE CALLED SAW.

SAW
X
XX
XXX
XXXX
XXX
XX
X

X
XX
XXX
XXXX
XXX
XX
X

X
XX
XXX
XXXX
XXX
XX
X

HERE IS A LISTING OF SAW.
LIST SAW

TO SAW
10 PRINT "X"
20 PRINT "XX"
30 PRINT "XXX"
40 PRINT "XXXX"
50 PRINT "XXX"
60 PRINT "XX"
70 PRINT "X"
80 PRINT ""
90 PRINT "X"
100 PRINT "XX"
110 PRINT "XXX"
120 PRINT "XXXX"
130 PRINT "XXX"
140 PRINT "XX"
150 PRINT "X"
160 PRINT ""
170 PRINT "X"
180 PRINT "XX"
190 PRINT "XXX"
```

```
200 PRINT "XXXX"
210 PRINT "XXX"
220 PRINT "XX"
230 PRINT "X"
END


QUESTION: WHICH LINES WOULD YOU ERASE SO THAT THE DESIGN WILL BE JOINED
TOGETHER LIKE A REAL SAW?
*80 160
THANKS AGAIN. YOU ARE A GREAT HELP.

HERE IS THE THIRD PROCEDURE. IT IS CALLED ALPHABETICAL.

ALPHABETICAL
ALAN
BILL
DAVID
GEORGE
FRED

LIST ALPHABETICAL

TO ALPHABETICAL
10 PRINT "ALAN"
20 PRINT "BILL"
30 PRINT "DAVID"
40 PRINT "GEORGE"
50 PRINT "FRED"
END


I HAVE THREE QUESTIONS ABOUT THIS PROCEDURE. THE FIRST IS -- WHICH LINE
IS OUT OF ORDER?
*50
THAT'S GOOD, THANKS.

QUESTION 2: WHAT NUMBER WOULD YOU GIVE THAT LINE TO PUT IT IN THE RIGHT
PLACE?
*39
YOU'RE SO RIGHT. HERE'S QUESTION 3: WHAT NUMBER WOULD YOU GIVE TO THE
INSTRUCTION - PRINT "CHARLES"
*21

THANK YOU FOR YOUR HELP. YOU COULD HELP ME EVEN MORE IF YOU HAVE TIME,
BY MAKING THE CORRECTIONS FOR ME IN EACH OF THE PROCEDURES.
←
```

```
←REVIEW

HERE IS A PROCEDURE WITH SEVERAL INSTRUCTIONS. THE PROCEDURE IS CALLED
- REVIEW-1. @

I WILL LIST THE PROCEDURE - REVIEW-1 - FOR YOU, WHEN YOU ARE READY. @


TO REVIEW-1
10 PRINT "COMPUTER"
20 PRINT "LOGO IS A COMPUTER LANGUAGE"
30 PRINT FIRST OF "COMPUTER"
40 PRINT LAST OF "COMPUTER"
50 PRINT BUTFIRST OF "COMPUTER"
60 PRINT BUTLAST OF "COMPUTER"
70 PRINT FIRST OF "COMPUTER LANGUAGE"
80 PRINT LAST OF "COMPUTER LANGUAGE"
END


PRETEND YOU ARE THE COMPUTER. I WILL ASK YOU A QUESTION ABOUT EACH
LINE. TELL ME WHAT YOU WOULD ANSWER. @

READY? HERE WE GO.
WHAT SHOULD THE COMPUTER ANSWER WHEN IT READS LINE 10?
*COMPUTER

GOOD. NOW WHAT WOULD IT DO WITH LINE 20?
*LOGO IS A COMPUTER LANGUAGE

GREAT! NOW, HOW ABOUT LINE 30?
*C

WHAT WOULD YOU DO WITH LINE 40?
*R

GOOD FOR YOU! NOW TRY LINE 50.
*OMPUTER
```

```
VERY GOOD. LOOK NOW AT LINE 60.
*P
SOMETHING IS NOT RIGHT. TRY AGAIN.
*COMPUTE

NOW FOR LINE 70
*COMPUTER

HOW ABOUT THE LAST ONE, LINE 80?
*LANGUAGE

NOW THAT YOU KNOW WHAT EACH LINE OF THE PROCEDURE SHOULD DO, TRY OUT
THE PROCEDURE AND SEE WHAT HAPPENS. IN ORDER TO TRY OUT THIS PROCEDURE,
JUST TYPE ITS NAME.
*LOGO
THAT DOESN'T SEEM TO BE QUITE RIGHT. TRY AGAIN.
*REVIEW-1
COMPUTER
LOGO IS A COMPUTER LANGUAGE
C
R
OMPUTER
COMPUTE
COMPUTER
LANGUAGE

IF YOU HAVE MORE TIME, TRY ADDING A FEW LINES TO THIS PROCEDURE.
←


 EDIT REVIEW-1
>90 PRIM\NT FIRST OF LOGOWW\\\\\\\"LOGO"
>100 PRINT BUTFIRST OF LOGO\\\\"LOGO"
>END
REVIEW-1 DEFINED
←REVIEW-1
COMPUTER
LOGO IS A COMPUTER LANGUAGE
C
R
OMPUTER
COMPUTE
COMPUTER
LANGUAGE
L
OGO
←
```

```
←LEARNDOUBLE   (LESSON TWELVE)

HERE IS A PROCEDURE WHICH WILL PRINT DOUBLE THE WORD - "CAT" @

WHEN YOU ARE READY, I WILL TRY THE PROCEDURE FOR YOU. THEN I WILL SHOW
IT TO YOU. @

DOUBLECAT
CATCAT

LIST DOUBLECAT

TO DOUBLECAT
10 PRINT WORD OF "CAT" AND "CAT"
END

@

NOW IT'S YOUR TURN. YOU CAN RUN DOUBLECAT YOURSELF IF YOU LIKE. THEN
TRY TO WRITE YOUR OWN PROCEDURE - DOUBLEDOG. @

WHEN YOU ARE SURE DOUBLEDOG WORKS, TYPE - MORE - AND YOU WILL GET MORE
INSTRUCTIONS.
←TO DOUBLEDOG
>20 PRINY\T WORD OF "DOG" AND "DOG"
>END
DOUBLEDOG DEFINED
←DOUBLEDOG
DOGDOG
←MORE
THIS TIME WRITE A PROCEDURE WHICH WILL DOUBLE "HA". YOU CAN CALL IT
DOUBLEHA IF YOU WISH. WHEN YOU KNOW IT WORKS, TYPE - MORE - AGAIN.
←TO DOUBLEHA
>70 PRINT WORD OF "HA" AND "HA"
>END
DOUBLEHA DEFINED
←DOUBLEHA
HAHA
←MORE
NOW WRITE A PROCEDURE WHICH WILL DOUBLE YOUR NAME. FOR EXAMPLE, IF MY
NAME WERE JOE, I WOULD CALL MY PROCEDURE DOUBLEJOE. WHEN YOU ARE SURE
YOUR PROCEDURE WORKS, TYPE - NEWIDEA.
←TO DOUBLERUTH-ANNE
>900 PRINT WORD OF "RUTH-ANNE" AND "RUTH-ANNE"
>END
DOUBLERUTH-ANNE DEFINED
←DOUBLERUTH-ANNE
RUTH-ANNERUTH-ANNE
```

```
←NEWIDEA

DOESN'T IT SEEM LIKE A LOT OF WORK TO HAVE TO WRITE A NEW PROCEDURE
EVERY TIME WE WANT TO DOUBLE A WORD? @

IT CERTAINLY WOULD BE EASIER IF WE HAD ONE PROCEDURE WHICH WOULD DOUBLE
ANY WORD WE GAVE IT. WE CAN WRITE ONE. SO LET'S! @

THE SECRET IS TO USE A NAME WHICH WILL STAND FOR ANY WORD WE WANT. @

HERE IS THE PROCEDURE:

LIST DOUBLE

TO DOUBLE /ANYWORD/
10 PRINT WORD OF /ANYWORD/ AND /ANYWORD/
END


NOW YOU TYPE - DOUBLE "CAT"
*DOUBLE "CAT"
CATCAT

WHEREVER /ANYWORD/ APPEARS IN THE PROCEDURE, "ANYWORD" BECOMES THE NAME
OF "CAT"

NOW TRY DOUBLE WITH LOTS OF OTHER WORDS AND SEE WHAT HAPPENS. WHEN YOU
ARE READY TO TRY SOMETHING NEW, TYPE - SUGGESTION.
←DOUBLE "HE IS A NICE BOY"
YOU CAN'T MAKE A WORD OUT OF A SENTENCE. (LOGO complains to the student)
I WAS AT LINE 10 IN DOUBLE.     (But, of course, he accepts the
←DOUBLE "HE I\\ISANICEBOY"      challenge.  He gives LOGO an accept-
HEISANICEBOYHEISANICEBOY        able word by squeezing together the
←SUGGESTION                     words in his sentence.)

TRY WRITING A NEW PROCEDURE CALLED DOUBLEFIRST. @

DOUBLEFIRST WILL DOUBLE THE FIRST LETTER OF THE WORD YOU GIVE IT AS
INPUT. @

FOR EXAMPLE, IF YOU TELL IT - DOUBLEFIRST "JOE" IT WILL REPLY - JJOE. @
←TO DOUBLEFIRST
>10 PRINT WORD OF FIRST OF /ANYWORD/ AND /ANYWORD/
>END
DOUBLEFIRST DEFINED
←DOUBLEFIRST OF "JIM"
JJIM
```

```
←USEINPUT    (LESSON THIRTEEN)
```

HERE IS A PROCEDURE YOU MAY HAVE SEEN BEFORE. I WILL LIST IT FOR YOU. @

```
TO JOHN
10 PRINT "JOHN IS MY FRIEND."
20 PRINT "JOHN GOES TO EMERSON SCHOOL."
30 PRINT "JOHN IS IN GRADE THREE."
40 PRINT "JOHN IS VERY SMART."
END
```

IN ORDER TO MAKE THE PROCEDURE - JOHN - WORK, ALL YOU MUST DO IS TYPE -JOHN. TRY IT.

```
*JOHN
JOHN IS MY FRIEND.
JOHN GOES TO EMERSON SCHOOL.
JOHN IS IN GRADE THREE.
JOHN IS VERY SMART.
```

HERE IS ANOTHER PROCEDURE CALLED - ADDON. I WILL LIST IT FOR YOU WHEN YOU ARE READY. @

```
TO ADDON /WHO/
10 PRINT SENTENCE OF /WHO/ AND "IS MY FRIEND."
20 PRINT SENTENCE OF /WHO/ AND "GOES TO EMERSON SCHOOL."
30 PRINT SENTENCE OF /WHO/ AND "IS IN GRADE THREE"
40 PRINT SENTENCE OF /WHO/ AND "IS VERY SMART"
END
```

IN ORDER TO MAKE THIS PROCEDURE WORK, YOU MUST TYPE IN THE NAME OF THE PROCEDURE AND SOMETHING ELSE. @

YOU MUST TELL THE COMPUTER WHAT /WHO/ IS. YOU MUST TELL IT WHAT THING THE NAME "WHO" STANDS FOR. @

FOR EXAMPLE, IF YOU WANT THE NAME "WHO" TO STAND FOR "JOHN" THEN YOU MUST TYPE -- ADDON "JOHN"

YOU TYPE -- ADDON "JOHN" -- AND SEE WHAT HAPPENS.

```
*ADDON "JOHN"
JOHN IS MY FRIEND.
JOHN GOES TO EMERSON SCHOOL.
JOHN IS IN GRADE THREE
JOHN IS VERY SMART
```

NOW TRY ANOTHER INSTRUCTION OF YOUR OWN USING ADDON AND ANY INPUT YOU WANT .

```
*ADDON "TOMMY"
TOMMY IS MY FRIEND.
TOMMY GOES TO EMERSON SCHOOL.
TOMMY IS IN GRADE THREE
TOMMY IS VERY SMART


TRY ONE MORE.

*ADDON "RUTH"
RUTH IS MY FRIEND.
RUTH GOES TO EMERSON SCHOOL.
RUTH IS IN GRADE THREE
RUTH IS VERY SMART

NOW WE HAVE LOOKED AT TWO PROCEDURES.
THE FIRST ONE - JOHN - HAS NO INPUTS. WE JUST TYPE THE NAME OF THE
PROCEDURE TO MAKE IT WORK.
THE SECOND ONE - ADDON - TAKES ONE INPUT. WE MUST TELL THE COMPUTER
WHAT THE NAME STANDS FOR.
NOW LET US LOOK AT A THIRD PROCEDURE. @

TO ADDMORE /WHO/ AND /WHERE/
10 PRINT SENTENCE OF /WHO/ AND "IS MY FRIEND."
20 PRINT SENTENCE OF /WHO/ AND SENTENCE OF "GOES TO" AND /WHERE/
30 PRINT SENTENCE OF /WHO/ AND "IS IN GRADE THREE."
40 PRINT SENTENCE OF /WHO/ AND "IS VERY SMART."
END


YOU CAN SEE THAT THIS PROCEDURE NEEDS TWO INPUTS. YOU MUST TELL IT WHAT
"WHO" STANDS FOR AND WHAT "WHERE" STANDS FOR. @

IN ORDER TO MAKE THIS PROCEDURE WORK, YOU MUST GIVE IT AN INSTRUCTION
LIKE THIS -- ADDMORE "JOHN" "ANGIER SCHOOL"
YOU TRY THIS AND SEE WHAT HAPPENS.


*ADDMORE "JOHN" "ANGIER SCOO\\HOOL"
JOHN IS MY FRIEND.
JOHN GOES TO ANGIER SCHOOL
JOHN IS IN GRADE THREE.
JOHN IS VERY SMART.

WHEN YOU SEE THE ARROW, TRY SOME OTHER INPUTS WITH THIS PROCEDURE. @
```

```
←PRRT    (LESSON FOURTEEN)

THERE ARE TWO WAYS TO WRITE THE PROCEDURE - DOUBLE. WE HAVE ALREADY
WORKED WITH ONE OF THESE WAYS. HERE IT IS. @


TO DOUBLE-1 /ANYWORD/
10 PRINT WORD OF /ANYWORD/ AND /ANYWORD/
END


TEST DOUBLE-1 TO SEE IF IT WORKS.
*DOUBLE-1 "SET"
SETSET

NOW LOOK AT ANOTHER PROCEDURE WHICH WILL DO WHAT DOUBLE DOES, WHEN YOU
GIVE IT THE RIGHT INSTRUCTION. @


TO DOUBLE-2 /ANYWORD/
10 RETURN WORD OF /ANYWORD/ AND /ANYWORD/
END


LOOK VERY CAREFULLY AT THIS PROCEDURE TO FIND OUT HOW IT IS DIFFERENT
FROM DOUBLE-1. @

DID YOU NOTICE THAT DOUBLE-1 SAYS 'PRINT' WHILE DOUBLE-2 SAYS 'RETURN'?
'RETURN' TELLS THE COMPUTER - BRING SOMETHING BACK TO ME -- BUT IT DOES
NOT TELL THE COMPUTER TO WRITE ANYTHING. @

HERE ARE SOME EXAMPLES OF HOW THE WORD 'RETURN' MAKES A DIFFERENCE . @

YOU TYPE -- DOUBLE-2 "BOOK"
*DOUBLE-2 "BOOK"

YOU SAW THAT THE COMPUTER DID NOT PRINT A REPLY. IT DID RETURN AN
ANSWER WHICH YOU CANNOT SEE. @

WE CAN GET IT TO PRINT THIS ANSWER BY TYPING - PRINT DOUBLE-2 "BOOK" .
TRY THIS AND SEE.
*PRINT DOUBLE-2 "BOOK"
BOOKBOOK
```

```
NOW YOU CAN SEE THAT DOUBLE-1 AND DOUBLE-2 WILL DO THE SAME THING.
HOWEVER YOU MUST ASK THE COMPUTER TO PRINT DOUBLE-2 TO GET IT TO WRITE
OUT AN ANSWER. DOUBLE-1 HAS THE PRINT COMMAND IN ITS PROCEDURE, SO YOU
DO NOT HAVE TO ASK IT PRINT. @

YOU MUST BE WONDERING WHY WE SHOULD BOTHER TO USE 'RETURN' AT ALL. IT
DOES SEEM EASIER TO WRITE PROCEDURES LIKE DOUBLE-1. WHEN YOU DO THE
NEXT LESSON, YOU WILL SEE WHAT THE DIFFERENCE IS, AND WHY WE NEED TO
USE 'RETURN' MOST OF THE TIME, RATHER THAN 'PRINT' IN OUR PROCEDURES.
←
```

```
←DOUBLE-1 DOUBLE-1 "DOG"
DOGDOG
```

(This didn't work as expected - the procedure DOUBLE-1 does not return its answer - it merely prints it.)

```
←PRINT DOUBLE-2 DOUBLE-2 "DOG"
DOGDOGDOGDOG
←
```

(But this one works! The procedure DOUBLE-2 when performed on "DOG" returns its answer "DOGDOG" as input to the first DOUBLE-2, which returns its answer, "DOGDOGDOGDOG", to PRINT which then prints it.)

```
←USE  (LESSON FIFTEEN)
I AM GOING TO SHOW YOU SOME PROCEDURES AND ASK YOU TO PRETEND TO BE THE
COMPUTER. @

HERE IS THE FIRST PROCEDURE.

LIST MYSTERY-1

TO MYSTERY-1 /ANYWORD/
10 PRINT WORD OF BUTFIRST OF /ANYWORD/ AND FIRST OF /ANYWORD/
END


IF YOU WERE THE COMPUTER, WHAT WOULD YOU ANSWER IF YOU WERE GIVEN THIS
INSTRUCTION:
MYSTERY-1 "SOAK"
*SOAK
PLEASE TRY AGAIN.
*OAKS

GREAT! HERE IS THE NEXT QUESTION. WHAT WOULD YOU ANSWER TO THIS
INSTRUCTION:
MYSTERY-1 312"
*123

RIGHT! HERE IS THE LAST QUESTION ABOUT THIS PROCEDURE. WHAT WOULD YOU
ANSWER?
MYSTERY-1 "EAR"
*ARE

GOOD FOR YOU. NOW LET'S LOOK AT ANOTHER PROCEDURE.

LIST MYSTERY-2

TO MYSTERY-2 /FIRST WORD/ AND /SECOND WORD/
10 PRINT WORD OF /SECOND WORD/ AND /FIRST WORD/
END


THE FIRST QUESTION:
MYSTERY-2 "AT" AND "BE"
*BEAT

RIGHT! NEXT QUESTION:
MYSTERY-2 "12" AND "24"
*2412
```

```
YES! NOW TRY THIS ONE:
MYSTERY-2 "EN" AND "D"
*DEN

GOOD! NOW HERE IS THE THIRD PROCEDURE:

LIST MYSTERY-3

TO MYSTERY-3 /FIRST NUMBER/ AND /SECOND NUMBER/ AND /SIGN/
10 PRINT WORD OF /FIRST NUMBER/ AND WORD OF /SIGN/ AND /SECOND NUMBER/
END


FIRST QUESTION:
MYSTERY-3 "5" AND "9" AND "+"
*5+9

GREAT! SECOND QUESTION:
MYSTERY-3 "6" AND "12" AND "-"
*6-12

RIGHT! HERE IS THE LAST QUESTION:
MYSTERY-3 "1" AND "2" AND "/"
*1/2

NOW THAT YOU HAVE DONE SO WELL ON THIS GROUP, TYPE -- NEXT -- TO GET
ANOTHER SET OF MYSTERY PROCEDURES.
←NEXT

I HAVE THREE MYSTERY PROCEDURES -- MYSTERY-4, MYSTERY-5, AND MYSTERY-6.
@

FIRST YOU ARE TO LIST MYSTERY-4. @

THEN YOU ARE TO TRY MYSTERY-4 WITH YOUR OWN INPUTS. @

WHEN YOU ARE SURE THAT YOU CAN GET MYSTERY-4 TO WORK WITHOUT ANY
COMPLAINTS FROM THE COMPUTER, DO THE SAME THING WITH MYSTERY-5. @

FINALLY, WHEN MYSTERY-5 WORKS, DO THE SAME THING WITH MYSTERY-6. @

GOOD LUCK! WHEN YOU HAVE FINISHED, TYPE -- SURPRISE -- FOR SOME FUN.
←SURPRISE
I HAVE FIVE MORE SHORT MYSTERY PROCEDURES FOR YOU. THEY ARE CALLED
SURPRISE-1, SURPRISE-2, SURPRISE-3, SURPRISE-4, AND SURPRISE-5. @

I HAVE HIDDEN THEM SO YOU CANNOT SEE THE INSTRUCTIONS OF EACH
PROCEDURE. BUT I WILL LET YOU LOOK AT THE TITLES. @
```

```
TO SEE WHAT EACH SURPRISE PROCEDURE WILL DO, YOU MUST FIRST LIST THE
PROCEDURE. JUST TYPE, FOR EXAMPLE, -- LIST SURPRISE-1.@

THEN TRY OUT PROCEDURE - SURPRISE-1 - WITH THE RIGHT INSTRUCTIONS, AND
YOU WILL SEE WHAT IT DOES. @

AFTER SURPRISE-1 WORKS FOR YOU, TRY SURPRISE-2, SURPRISE-3, SURPRISE-4,
AND SURPRISE-5. HAVE FUN!
←LIST SURPRISE-1

TO SURPRISE-1 /BOY'S NAME/ AND /GIRL'S NAME/
```

(only the title line, which lists the input names, is visible)

```
←SURPRISE-1 "LISA" "TONY"
LISA LIKES TONY
←LIST SURPRISE-2

TO SURPRISE-2 /YOUR FIRST NAME/ AND /ONE KIND OF FOOD/


←SURPRISE-2 "JAY" "PIZZA"
JAY'S FAVORITE FOOD IS PIZZA
←SURPRISE-2 "JAY" "CHINESE FOOD"
JAY'S FAVORITE FOOD IS CHINESE FOOD
←LIST SURPRISE-3

TO SURPRISE-3


←SURPRISE-3
THE TIME IS NOW 1:48 PM
←LIST SURPRISE-4

TO SURPRISE-4 /YOUR LAST NAME/


←SURPRISE-4 "BORGES"
SEGRBO
←LIST SURPRISE-5
```

(It's not at all likely, from this single trial of SURPRISE-4 that the student could "see what it does")

```
TO SURPRISE-5 /FIRST NUMBER/ AND /SECOND NUMBER/ AND /THIRD NUMBER/


←SURPRISE-5 "1" "7" "5"
100 + 70 + 5 = 175
←
```

(A little more likely here.)

```
←LIST MYSTERY-4

TO MYSTERY-4 /FIRST NAME/ AND /MIDDLE NAME/ AND /LAST NAME/
10 PRINT WORD OF FIRST OF /FIRST NAME/ AND WORD OF FIRST OF /MIDDLE
   NAME/ AND FIRST OF /LAST NAME/
END

←MYSTERY-4 M\"MARY" AND "JAYE" AND "SIMMS"
MJS
←MYSTERY-4 "STEVEN" "HOWARD" "EPSTEIN"
SHE
←LIST MYSTERY-5

TO MYSTERY-5 /NUMBER/
10 PRINT DIFFERENCE OF /NUMBER/ AND /NUMBER/
END

←MYSTERY-5 "13"
0
←MYSTERY-5 "12345600"
0

←LIST MYSTERY-6

TO MYSTERY-6 /BIRTH MONTH/ AND /DAY/ AND /AGE/
10 PRINT SENTENCE OF "MY BIRTHDAY IS" AND SENTENCE OF /BIRTH MONTH/ AND
   /DAY/
20 PRINT SENTENCE OF "I AM" AND SENTENCE OF /AGE/ AND "YEARS OLD."
END

←MYSTERY-6 "JUNE" "5" "8"
MY BIRTHDAY IS JUNE 5
I AM 8 YEARS OLD.
←MYSTERY-6 "MAY" "1" "190"
MY BIRTHDAY IS MAY 1
I AM 190 YEARS OLD.


←MYSTERY-6 "EAT2" "123" "1000000000"
MY BIRTHDAY IS EAT2 123
I AM 1000000000 YEARS OLD.
```

billion million 10000 1000 1

(How does one say that number? The teacher helps.)

```
←SP  (LESSON SIXTEEN)

HERE ARE SOME SUGGESTIONS FOR PROCEDURES WHICH YOU MIGHT WRITE. @

THE FIRST PROCEDURE IS ANOTHER KIND OF DOUBLE PROCEDURE. THIS
PROCEDURES IS CALLED 'DBL' AND WILL DOUBLE ANY NUMBER. @

HERE IS AN EXAMPLE OF HOW DBL WORKS. YOU TYPE -- DBL "4"
*DBL "4"
8

NOW TRY DBL AGAIN BY TYPING -- DBL AND ANY NUMBER YOU CHOOSE.
*DBL "25"
50

NOW I HAVE ERASED MY PROCEDURE - DBL. WHEN YOU SEE THE ARROW, IT IS
YOUR TURN TO WRITE A PROCEDURE - DBL - WHICH WILL DO WHAT MY DBL DOES.
@

WHEN YOU ARE SURE YOUR DBL WORKS, TYPE -- NEXT - TO SEE MY NEXT
PROCEDURE.

←LIST DBL                  (It was erased!)
DBL ISN'T DEFINED.
←TO DBL ANY NUMBER
YOU NEED / MARKS AROUND EACH ARGUMENT.


←TO DBL /ANY NUMBER/
>10 PRINT SUM OF /ANY NUMBER/ AND /ANY NUMBER/
>END

←DBL 14
28
←DBL "5"
10
←DBL "345"
690
←DBL "1234567890"
2469135780
←DBL "0987654321"
1975308642
←NEXT
```

```
THE NEXT PROCEDURE IS CALLED SECOND. IT PRINTS THE SECOND LETTER OF ANY
WORD YOU GIVE IT. @

TRY OUT MY PROCEDURE SECOND. YOU TYPE -- SECOND "JUMP"
*SECOND "JUMP"
U

HERE IS ANOTHER CHANCE TO TRY OUT MY PROCEDURE SECOND. TYPE -- SECOND
"HELLO"
*SECOND "HELLO"
E

SECOND HAS NOW BEEN ERASED. IT'S YOUR TURN TO WRITE A PROCEDURE CALLED
SECOND WHICH WILL PRINT THE SECOND LETTER OF ANY WORD. HAVE FUN! @

WHEN YOU ARE SURE SECOND WORKS, TYPE -- STILLMORE- FOR THE LAST
PROCEDURE.
←TO SECOND /ANYWORD/
>10 PRINT SECOND LETTER OF /ANYWORD/   (His intention was clear -)
>END
SECOND DEFINED
←SECOND "ME"
LETTER ISN'T DEFINED.                  (but not to LOGO!)
I WAS AT LINE 10 IN SECOND.
←EDIT SECOND                           (So he edits the procedure
#10                                    by giving a valid instruction.)
>10 PRINT FIRST OF BUTFIRST OF /ANYWORD/
>END
SECOND DEFINED
←SECOND "ME"
E
←SECOND "SUPERCALAFRAGEALISTICEXPIALADOCHES"       (Wow!)
U
←SECOND "T"
                             (The answer is the Empty Word.)
←STILLMORE

HERE IS THE LAST PROCEDURE FOR YOU TO TRY. IT IS CALLED 'ADD.' ADD
TAKES THREE DIFFERENT NUMBERS AND ADDS THEM TOGETHER. @

HERE IS AN EXAMPLE OF HOW ADD WORKS. YOU TYPE - ADD "3" "6" "8"
*ADD "3" "6" "8"
17
```

```
HERE IS ANOTHER CHANCE TO TRY MY PROCEDURE -- ADD. TYPE -- ADD AND ANY
THREE NUMBERS YOU CHOOSE.
*ADD "43" "0" "43"
86

NOW IT'S YOUR TURN AGAIN. ADD IS GONE. PLEASE WRITE ONE OF YOUR OWN. @

WHEN YOU ARE SURE YOUR ADD WORKS, YOU ARE WELL ON YOUR WAY TO BECOMING
A CRACKERJACK PROGRAMMER.!
←TO ADD /ANYNUMBER/ AND /ANYNUMBER/ AND /ANYNUMBER/  (A beauty of a bug!)
>10 PRINT SUM OF SUM OF /ANYNUMBER/ AND /ANYNUMBER/ AND /ANYNUMBER/
>END
ADD DEFINED
←ADD "7" "7" "1"
21                                        (What's wrong?)
←ADD "1" "1" "1"
3                                         (It worked this time!)
←ADD "12" "12" "1"
36                                        (But again it failed.)
#EXAMPLE                                  (The teacher intervened here to help
←TO TEST /FIRSTWORD/ AND /SECONDWORD/       explain the difficulty with this
>10 PRINT WORD OF /FIRSTWORD/ AND /SECONDWORD/      example.)
>END
TEST DEFINED                              (Let's write a procedure called TEST)


                                          (Here is another way of writing TEST)
←     TO TEST-1 /FIRSTWORD/ AND /FIRSTWORD/
>10 PRINT WORD OF /FIRSTWORD / AND /S\FIRSTT\WORD/
>END
TEST-1 DEFINED                            (Now let's try the two procedures)
←TEST "DOG" "CAT"
DOGCAT                                    (Right!)
←TEST-1   "DOG" "CAT"
DOGDOG                                    (Wrong!)  (The error is clear - now
←TO  ADD /FIRSTWORD/ AND /SECONDWORD/ AND /THIRDWORD/  we can fix ADD.)
ADD IS ALREADY DEFINED.
←ERASE ADD                                (But first we must clear the slate.)
←TO ADD /FIRSTWORD/ AND /SECONDWORD/ AND /THIRDWORD/
>10  PRINT SUM OF SUM OF /FIRSTWORD/ AND /SECONDWORD/ AND /THIRDWORD/
>END
ADD DEFINED
←ADD "3" "2" "1"
6                                         (Good!)
←
```

```
←TEACHIF  (LESSON SEVENTEEN)

IN LOGO WE CAN ASK THE COMPUTER TO DO DIFFERENT KINDS OF WORK. @

WE CAN ASK IT TO TYPE SOMETHING, BY SAYING PRINT. WE CAN ASK IT TO NAME
SOMETHING BY SAYING CALL. @

NOW WE ARE GOING TO LEARN HOW TO ASK IT TO CHECK SOMETHING. @

SUPPOSE WE WANTED TO FIND OUT IF 5+3=8. WE CAN ASK THE COMPUTER IN
LOGO. @

WE DO THIS BY WRITING TWO INSTRUCTIONS ON TWO SEPARATE LINES. @

THE FIRST INSTRUCTION WOULD BE -- IS SUM OF "5" AND "3" "8"

THE SECOND INSTRUCTION MIGHT BE -- IF YES PRINT "YES"

YOU TYPE IN THE FIRST INSTRUCTION.
*IS SUM OF "5" AND "#\3" "8"
NOW TYPE THE SECOND INSTRUCTION.
*IF YES PRINT "YES"
YES

HERE ARE TWO MORE INSTRUCTIONS FOR YOU TO TYPE: --
 THE FIRST ONE IS: -- IS FIRST OF "TREE" "T"
 THE SECOND ONE IS: -- IF YES PRINT "YES"
*IS FIRST OF TREE "T"
PLEASE TRY AGAIN.
*IS FIRST OF "TREE" "T"
*IF YES PRINT "YES"
YES

HERE IS THE THIRD SET OF INSTRUCTIONS FOR YOU TO TRY: --
 IS COUNT OF "DOG" "4"
 IF YES PRINT "YES"
*IS COUNT OF "DOG" "4"
*IF YES PRINT 2Y\\"YES"

WHAT HAPPENED? DID THE COMPUTER RETURN EMPTY? DO YOU KNOW WHY? @

IT RETURNED EMPTY BECAUSE IT WAS TOLD TO PRINT YES ONLY IF THE ANSWER
WAS YES. SINCE THE ANSWER WAS NOT YES, IT RETURNED EMPTY. @
```

```
HERE IS THE LAST SET OF INSTRUCTIONS FOR YOU TO TRY: --
 IS LAST OF "TREE" "T"
 IF NO PRINT "NO"
*IS LAST OF T\"TREE" "T"
*IF NO PRINT "NO"
NO

IN THIS LAST SET OF INSTRUCTIONS, WE USED - IF NO -- INSTEAD OF - IF
YES. IF NO AND IF YES WORK IN THE SAME WAY.@

LOOK BACK AT THE DIFFERENT INSTRUCTIONS WE HAVE GIVEN THE COMPUTER
USING - IS. IS TESTS TWO THINGS TO FIND OUT IF THEY ARE THE SAME. @

I AM GOING TO ASK YOU TO TRY SOME 'IS' INSTRUCTIONS OF YOUR OWN, BUT I
WILL ALSO GIVE YOU A LIST OF IDEAS THAT YOU CAN WORK WITH. @

HERE IS THE LIST:

IS "LOGO" "LOGO"
IS DIFF OF "4" AND "1" "1"
IS WORD OF "CAT" AND "DOG" "CAT DOG"
IS "THIS SCHOOL" "EMERSON"
IS "BLACK" "WHITE"
IS BUTFIRST OF BUTFIRST OF "GAME" "ME"
IS /BLUE/ /BLUE/
IS /GREEN/ /BLUE/
IS /EMPTY/ ""
IS BUTFIRST OF BUTLAST OF "BYE" "E"
←IS /SNOOPY/ A /DOG/
A ISN'T DEFINED.
←IS "LOGO" "LOGO"
←IF YES PRINT "OF CORSE DUW\MMY EVERYBODY KNOWS THAT"
OF CORSE DUMMY EVERYBODY KNOWS THAT
←IS DIFF OF "4" AND2\ "1" "1"
←IF NO PRINT "THAT IS SO HARD I  DO NOT KNOW"
THAT IS SO HARD I DO NOT KNOW
←IS WORD OF "CAT" AND "DOG" "CAT DOG"
←IF NO PRINT "HOW SHOUD I KNOW"
HOW SHOUD I KNOW
←IS /GREEN//BLUE/
←IF YES PRINT "ONLY MARY JAYE WOULD NOT KNOW THAT"
ONLY MARY JAYE WOULD NOT KNOW THAT
←IS "THIS SCHOOL!\" "EMERSON"
←IF YES PRINT " "
←IF NO PRINT "I THIN\\\\DON\ NOT THINK SO"
I DO NOT THINK SO
```

```
←START   (LESSON EIGHTEEN)

HERE IS A PROCEDURE WE USED WHEN WE FIRST STARTED WORKING WITH THE
COMPUTER, @

FIRST ASK THE COMPUTER TO -- LIST SAYHI. @

THEN TRY OUT SAYHI TO SEE WHAT IT DOES. @

WHEN YOU ARE READY TO GO ON, TYPE -- MORE.
←LIST SAYHI

TO SAYHI
10 PRINT "HI"
20 SAYHI
END
←SAYHI
HI
HI
HI
HI
HI
 .
 :     (The program was stopped by hitting the BREAK key.)


←MORE

NOW IT'S YOUR TURN TO WRITE A PROCEDURE. @

TRY TO WRITE A PROCEDURE WHICH WILL PRINT OUT YOUR NAME JUST THE WAY
SAYHI PRINTED HI. @

WHEN IT WORKS, TYPE -- NEXT -- FOR ANOTHER SUGGESTION.
←TO SAYRUTH-ANNE
>10 PRINT "RUTH-ANNE"
>20 SAY \RUTH-ANNE
>END
SAYRUTH-ANNE DEFINED
←SAYRUTH-ANNE
RUTH-ANNE
RUTH-ANNE
RUTH-ANNE
RUTH-ANNE
RUTH-ANNE
  :    :
```

```
←NEXT

NOW ARE YOU READY FOR THE CHALLENGE? ANSWER YES OR NO.
*YES
THIS TIME TRY TO WRITE A PROCEDURE THAT WILL WORK ON ANY WORD YOU GIVE
IT. @

PLEASE CALL YOUR PROCEDURE -- WORK. IF I USED YOUR PROCEDURE AND SAID
WORK "MARGE" IT WOULD TYPE -- MARGE -- RIGHT DOWN THE PAGE. @

HOW ABOUT IT? ARE YOU GAME? GOOD LUCK!
```

```
←TO SAY /ANYWORD/
>10 PRINT /ANYWORD/
>20 SAY /ANYWORD/
>END
SAY DEFINED
```

(It apparently seemed more natural to call her procedure SAY.)

```
←SAY "I WILL MISS YOU"
I WILL MISS YOU
I WILL MISS YOU
I WILL MISS YOU
I WILL MISS YOU
I WILL MISS YOU
I WILL MISS YOU
I WILL MISS YOU
    :   :
```

(A last day farewell from the child to the teacher.)

Examples of LOGO Lesson Programs

The next two typescripts are the LOGO programs for Lessons One and Eighteen. These two programs are typical of the series of twenty. They are included to illustrate the ease with which LOGO can be used to express teaching interactions like those shown in the preceding pages. The programs were written by the teacher, Mrs. Bloom, who had not done any programming prior to learning LOGO in the summer of 1968.

The program for Lesson One is composed of three procedures - SKIP, REQ, and LESSON1. Their operation is as follows. SKIP /N/ causes the teletype to skip /N/ lines by printing the EMPTY word /N/ times. REQ first executes a request instruction, i.e., waits for a user to type in a message, and then skips a line. The lesson begins when the student calls the main procedure LESSON1, which types a series of messages back to him. The messages are punctuated by SKIPs. The student calls forth each succeeding message when he is ready by hitting the carriage return key, thus completing a REQ.

The instructions in lines 100, 160, and 200 refer to the names /C1/, /C2/, and /INSTRUCTION/ (listed at the end of the program), and illustrate the printout of messages containing embedded quotes.

LESSON ONE

```
TO SKIP /NUMBER/
10 TEST IS /NUMBER/ "0"
20 IF TRUE STOP
30 PRINT /EMPTY/
40 SKIP DIFFERENCE /NUMBER/ "1"
END

TO REQ
10 REQUEST
20 SKIP "1"
END

TO LESSON1
10 SKIP "1"
20 TYPE "IN THIS LESSON THERE ARE SOME LINES TO READ. THE COMPUTER
   TYPES VERY FAST. IT WILL STOP EVERY SO OFTEN SO THAT YOU WILL HAVE
   TIME TO READ. WHEN YOU WANT IT TO GO ON, PRESS THE RETURN KEY. @"
30 REQ
40 TYPE "THIS MARK @ MEANS THAT THE COMPUTER HAS MORE TO TELL YOU.
   PRESS THE RETURN KEY WHEN YOU ARE READY TO READ IT. @"
50 REQ
60 TYPE "THE COMPUTER UNDERSTANDS SOME SPECIAL COMMANDS. THE FIRST ONE
   WE WILL TALK ABOUT IS -- PRINT. @"
70 REQ
80 PRINT "LET'S TELL THE COMPUTER TO PRINT A WORD. I WILL TYPE THE
   COMMAND. YOU PRESS THE RETURN KEY WHEN I AM FINISHED. @"
90 SKIP "1"
100 TYPE /C1/
110 REQUEST
120 DO /C1/
130 SKIP "1"
140 PRINT "NOW LET'S TELL THE COMPUTER TO PRINT A SENTENCE. YOU PRESS
    THE RETURN KEY THIS TIME TOO."
150 SKIP "1"
160 TYPE /C2/
170 REQUEST
180 DO /C2/
190 SKIP "1"
200 TYPE /INSTRUCTION/
210 REQ
220 TYPE "NOW TRY TO MAKE THE COMPUTER PRINT SOME OF YOUR OWN THINGS.
    @"
230 REQ
240 TYPE "DON'T WORRY ABOUT MAKING MISTAKES! WE ALL DO IT! THE COMPUTER
    WILL TRY TO HELP YOU BY EXPLAINING WHAT WENT WRONG. @"
250 REQ
260 PRINT "HAVE FUN!"
END
/C1/ IS "PRINT "CABBAGE""
/C2/ IS "PRINT "DO YOU LIKE THE BEATLES?""
/INSTRUCTION/ IS "NOTICE THAT THERE WERE QUOTATION (" ") MARKS AROUND
    THE EXACT WORDS THE COMPUTER WAS ASKED TO PRINT. @"
```

The program for Lesson Eighteen also uses the SKIP and REQ procedures (these are not reproduced again). The program comprises four main procedures - START, SAYHI, MORE, and NEXT - which are executed successively, as follows. START tells the student to list the procedure SAYHI, then to execute it to see what it does, and after that to type MORE. The procedure SAYHI prints out HI repetitively. When the student types MORE, he starts up the MORE procedure which asks the student to write a program for printing out his name, just as SAYHI printed HI, and then to type NEXT. When the student types NEXT, he starts up the NEXT procedure which asks him if he is ready for a challenge. If he answers YES (if his first answer is NO, he is asked to reconsider; if he is insistently negative, the lesson ends), he is given the problem of writing a SAY procedure that extends the preceding ones by printing out any given message repetitively.

Note the alternation of control between the teaching program and the student across successive phases of the interaction. In START, MORE, and NEXT the "teacher" is directing the student. But when using SAYHI, and when writing and using the two programs that are assigned to him, the student is using LOGO on his own. Embedding this kind of open-ended work enlivens the instruction and helps avoid the rigid, heavy-handed, stereotypy characteristic of much current computerized teaching.

LESSON EIGHTEEN

```
TO START
10 SKIP "1"
20 TYPE "HERE IS A PROCEDURE WE USED WHEN WE FIRST STARTED WORKING WITH
   THE COMPUTER. @"
30 REQ
40 TYPE "FIRST ASK THE COMPUTER TO -- LIST SAYHI. @"
50 REQ
60 TYPE "THEN TRY OUT SAYHI TO SEE WHAT IT DOES. @"
70 REQ
80 PRINT "WHEN YOU ARE READY TO GO ON, TYPE -- MORE."
END

TO MORE
10 SKIP "1"
20 TYPE "NOW IT'S YOUR TURN TO WRITE A PROCEDURE. @"
30 REQ
40 TYPE "TRY TO WRITE A PROCEDURE WHICH WILL PRINT OUT YOUR NAME JUST
   THE WAY SAYHI PRINTED HI. @"
50 REQ
60 PRINT "WHEN IT WORKS, TYPE -- NEXT -- FOR ANOTHER SUGGESTION."
END

TO NEXT
10 SKIP "1"
20 PRINT "NOW ARE YOU READY FOR THE CHALLENGE? ANSWER YES OR NO."
30 REQUEST "ANS"
40 IS /ANS/ "YES"
45 IF NO PRINT "AW, GEE. WON'T YOU PLEASE GIVE IT A TRY? ANSWER YES OR
   NO."
50 IF NO REQUEST "ANS"
60 IF NO IS /ANS/ "YES"
70 IF NO PRINT "OKAY, THEN. MAYBE YOU'LL FEEL MORE DARING LATER."
75 IF NO RETURN
80 TYPE "THIS TIME TRY TO WRITE A PROCEDURE THAT WILL WORK ON ANY WORD
   YOU GIVE IT. @"
90 REQ
100 TYPE SENTENCE SENTENCE "PLEASE CALL YOUR PROCEDURE -- WORK. IF I
   USED YOUR PROCEDURE AND SAID" /WM/ "IT WOULD TYPE -- MARGE -- RIGHT
   DOWN THE PAGE. @"
110 REQ
120 PRINT "HOW ABOUT IT? ARE YOU GAME? GOOD LUCK!"
END

TO SAYHI
10 PRINT "HI"
20 HI
END


/WM/ IS "WORK "MARGE""
```

3.4 The Games

The children's "work" on their lessons was punctuated from time to time by playing various games at the computer terminal. The games included:

Tic-Tac-Toe
Four-in-a-Row
Nim
Thirty-one
Wordhunt
Hangman
Guessword

Printout from children's play with Four-in-a-Row, Nim, Thirty-one, Wordhunt, and Hangman is reproduced in this section.

These games are described on the printouts. The other two games are similar to those included here. Thus, Tic-Tac-Toe (X's and O's) is the familiar 3 X 3 board game which might aptly be called Three-in-a-Row, and Guessword, like Hangman, is a word guessing game. Two other entertainments were provided by (1) a program SNOW which produces large signs on teletype paper, given the message text - either black on white or white on black, and (2) a LOGO program SNOOPY which makes a teletype drawing of the famous dog carrying a flag emblazoned with whatever name the child requests.

A slightly truncated SNOOPY picture is reproduced at the end of this section - the requested input was "LOGO".

```
←FOUR-IN-A-ROW
DO YOU KNOW HOW TO PLAY? ANSWER YES OR NO.
*NO


THIS GAME IS SIMILAR TO TIC-TAC-TOE IN MANY WAYS.
YOU WILL PLAY ON A BOARD THAT LOOKS LIKE THIS:

    0  1  2  3  4  5

 5  *  *  *  *  *  *  5

 4  *  *  *  *  *  *  4

 3  *  *  *  *  *  *  3

 2  *  *  *  *  *  *  2

 1  *  *  *  *  *  *  1

 0  *  *  *  *  *  *  0

    0  1  2  3  4  5

EACH * CAN BE NAMED BY TWO DIGITS. THE FIRST DIGIT COMES FROM THE
HORIZONTAL ROW OF NUMBERS. THE SECOND DIGIT COMES FROM THE VERTICAL
COLUMN OF NUMBERS.

FOR EXAMPLE, SUPPOSE WE WANT TO REPLACE THE ? IN THE BOARD BELOW WITH
A N X.


    0  1  2  3  4  5

 5  *  *  *  *  *  *  5

 4  *  *  *  *  *  *  4

 3  *  *  ?  *  *  *  3

 2  *  *  *  *  *  *  2

 1  *  *  *  *  *  *  1

 0  *  *  *  *  *  *  0

    0  1  2  3  4  5

THE QUESTION MARK IS AT 23, BECAUSE IT IS AT 2 ON THE HORIZONTAL ROW
AND AT 3 ON THE VERTICAL COLUMN.
```

```
ONE PLAYER WILL PLAY X'S. THE OTHER PLAYER WILL PLAY O'S.
EACH PLAYER TAKES TURNS PUTTING IN HIS MARKS.

IF YOU CHOOSE A POINT THAT HAS ALREADY BEEN TAKEN, YOU MUST LOSE YOUR
T URN.  SO SELECT YOUR POINTS CAREFULLY!
THE PURPOSE OF THE GAME, OF COURSE, IS TO GET FOUR X'S OR 4 O'S IN A
R OW. THE ROW MAY BE VERTICAL, HORIZONTAL, OR DIAGONAL.
THE COMPUTER WILL CHECK EACH TIME TO SEE IF THERE IS A WINNER.

WHO IS PLAYING WITH X'S? (TYPE IN YOUR NAME.)
* ROSEMARIE PHILLIPS
WHO IS PLAYING WITH O'S? (TYPE IN YOUR NAME.)
*MRS. BLOOM
ROSEMARIE PHILLIPS ,PLEASE TELL ME YOUR MOVE.
*51

     0  1  2  3  4  5

  5  *  *  *  *  *  *  5

  4  *  *  *  *  *  *  4

  3  *  *  *  *  *  *  3

  2  *  *  *  *  *  *  2

  1  *  *  *  *  *  X  1

  0  *  *  *  *  *  *  0

     0  1  2  3  4  5

MRS. BLOOM ,WHAT IS YOUR MOVE, PLEASE?
*42

     0  1  2  3  4  5

  5  *  *  *  *  *  *  5

  4  *  *  *  *  *  *  4

  3  *  *  *  *  *  *  3

  2  *  *  *  *  O  *  2

  1  *  *  *  *  *  X  1

  0  *  *  *  *  *  *  0

     0  1  2  3  4  5
```

```
        :   :

ROSEMARIE PHILLIPS ,PLEASE TELL ME YOUR MOVE.
*00

       0  1  2  3  4  5

    5  *  *  *  *  *  *  5

    4  *  *  *  *  *  *  4

    3  *  *  *  *  *  *  3

    2  *  *  X  O  O  O  2

    1  *  *  *  O  *  X  1

    0  X  *  O  X  X  X  0

       0  1  2  3  4  5

MRS. BLOOM ,WHAT IS YOUR MOVE, PLEASE?
*53

       0  1  2  3  4  5

    5  *  *  *  *  *  *  5

    4  *  *  *  *  *  *  4

    3  *  *  *  *  *  O  3

    2  *  *  X  O  O  O  2

    1  *  *  *  O  *  X  1

    0  X  *  O  X  X  X  0

       0  1  2  3  4  5

CONGRATULATIONS, MRS. BLOOM YOU'VE WON!
```

```
←NIM
DO YOU KNOW HOW TO PLAY NIM? PLEASE ANSWER YES OR NO.
*NO

TO PLAY THE GAME WE WILL NEED A STRING OF X'S THAT LOOKS LIKE THIS:
XXXXXXXXXX. I WILL LET YOU DECIDE HOW MANY X'S WE SHOULD USE.


AFTER YOU TELL ME HOW MANY X'S YOU WANT TO PLAY WITH, I WILL TYPE OUT
THE CORRECT AMOUNT. THEN YOU AND I WILL TAKE TURNS REMOVING 1, 2, OR 3
X'S AT A TIME. THE OBJECT OF THE GAME IS TO LEAVE 1 X. THE LOSER IS THE
ONE WHO MUST TAKE THE LAST X.


I'M ALL READY. ARE YOU?

YOU'RE LUCKY. YOU GET TO CHOOSE HOW MANY X'S WE ARE GOING TO PLAY WITH.
TELL ME HOW MANY X'S YOU WANT TO USE. THEN I WILL PRINT THEM SO THAT WE
CAN BOTH LOOK AT THEM. PLEASE TELL ME TO PRINT AT LEAST 6 X'S.
*19


XXXXXXXXXXXXXXXXXXX (19)


DO YOU WANT TO GO FIRST?
*YES
HOW MANY X'S DO YOU WANT TO REMOVE? YOU MAY REMOVE 1, 2, OR 3 X'S.
*3


XXXXXXXXXXXXXXXX (16)


I WILL REMOVE 3 X'S.


XXXXXXXXXXXXX (13)


HOW MANY X'S DO YOU WANT TO REMOVE? YOU MAY REMOVE 1, 2, OR 3 X'S.
*3


XXXXXXXXXX (10)
```

```
I WILL REMOVE 1 X'S.

XXXXXXXXX (9)

HOW MANY X'S DO YOU WANT TO REMOVE? YOU MAY REMOVE 1, 2, OR 3 X'S.
*4

YOUR REQUEST MUST BE 1,2,OR 3.
HOW MANY X'S DO YOU WANT TO REMOVE? YOU MAY REMOVE 1, 2, OR 3 X'S.
*2
XXXXXXX (7)

I WILL REMOVE 2 X'S.

XXXXX (5)

HOW MANY X'S DO YOU WANT TO REMOVE? YOU MAY REMOVE 1, 2, OR 3 X'S.
*2

XXX (3)

I WILL REMOVE 2 X'S.
X (1)

AHA! LOOKS LIKE YOU'VE LOST, BUT I'LL GIVE YOU ANOTHER CHANCE TO BEAT
ME, IF YOU'D LIKE.
DO YOU WANT TO PLAY AGAIN?
*YES
TELL ME HOW MANY X'S YOU WANT TO USE. THEN I WILL PRINT THEM SO THAT WE
CAN BOTH LOOK AT THEM. PLEASE TELL ME TO PRINT AT LEAST 6 X'S.
*15

XXXXXXXXXXXXXXX (15)

DO YOU WANT TO GO FIRST?
*YES
HOW MANY X'S DO YOU WANT TO REMOVE? YOU MAY REMOVE 1, 2, OR 3 X'S.
*2

XXXXXXXXXXXXX (13)
```

```
I WILL REMOVE 3 X'S.

XXXXXXXXXX (10)

HOW MANY X'S DO YOU WANT TO REMOVE? YOU MAY REMOVE 1, 2, OR 3 X'S.
*1

XXXXXXXXX (9)

I WILL REMOVE 1 X'S.

XXXXXXXX (8)

HOW MANY X'S DO YOU WANT TO REMOVE? YOU MAY REMOVE 1, 2, OR 3 X'S.
*3

XXXXX (5)

I WILL REMOVE 1 X'S.

XXXX (4)

HOW MANY X'S DO YOU WANT TO REMOVE? YOU MAY REMOVE 1, 2, OR 3 X'S.
*3

X (1)

CONGRATULATIONS! YOU WON! YOU MUST BE VERY SMART! BUT I'M NOT SO DUMB
EITHER. PLEASE GIVE ME ANOTHER CHANCE!

DO YOU WANT TO PLAY AGAIN?
*NO
GOODBYE FOR NOW. LET'S PLAY AGAIN SOMETIME.
←
```

```
-THIRTY ONE

DO YOU KNOW HOW TO PLAY?  ANSWER YES OR NO.
*NO

HERE ARE THE DIRECTIONS.

THE COMPUTER WILL ASK YOU FOR YOUR NUMBER.  YOUR NUMBER MUST BE 1, 2, 3,
4, 5, OR 6.

THEN THE COMPUTER WILL CHOOSE A NUMBER FROM 1 TO 6 AND ADD IT TO YOURS.

YOUR NEXT NUMBER WILL BE ADDED TO THE TOTAL, AND YOU AND THE COMPUTER
WILL TAKE TURNS SELECTING NUMBERS UNTIL ONE OF YOU IS ABLE TO REACH 31.
THE WINNER IS THE ONE WHO HITS THIRTY-ONE EXACTLY.


 DO YOU WANT TO GO FIRST.  ANSWER YES OR NO.
 *YES
 GIVE ME YOUR NUMBER, PLEASE.
 *5

 THE TOTAL IS NOW 5

 I WILL ADD 5

 THE TOTAL IS NOW 10

 GIVE ME YOUR NUMBER, PLEASE.
 *4

 THE TOTAL IS NOW 14

 I WILL ADD 3

 THE TOTAL IS NOW 17
 GIVE ME YOUR NUMBER, PLEASE.
 *1

 THE TOTAL IS NOW 18

 I WILL ADD 6

 THE TOTAL IS NOW 24
 GIVE ME YOUR NUMBER, PLEASE.
 *4

 THE TOTAL IS NOW 28
```

```
I WILL ADD 3

THE TOTAL IS NOW 31

AH HA! AREN'T I CLEVER?
DO YOU WANT TO PLAY AGAIN?
*YES

GIVE ME YOUR NUMBER, PLEASE.
*3

THE TOTAL IS NOW 3

I WILL ADD 4

THE TOTAL IS NOW 7

GIVE ME YOUR NUMBER, PLEASE.
*3

THE TOTAL IS NOW 10

I WILL ADD 2

THE TOTAL IS NOW 12

GIVE ME YOUR NUMBER, PLEASE.
*5

THE TOTAL IS NOW 17

I WILL ADD 1

THE TOTAL IS NOW 18

GIVE ME YOUR NUMBER, PLEASE.
*6

THE TOTAL IS NOW 24

I WILL ADD 6

THE TOTAL IS NOW 30

GIVE ME YOUR NUMBER, PLEASE.
*1

THE TOTAL IS NOW 31

CONGRATULATIONS, GENIUS!  YOU GOT ME!
```

```
←WORDHUNT

DO YOU KNOW HOW TO PLAY?
*NO

THINK ABOUT HOW MANY WORDS YOU CAN MAKE OUT OF THIS WORD -- TOWN. YOU
MAY USE SOME OR ALL OF THE LETTERS. @

I HAVE AN ANSWER LIST. I WILL GIVE YOU A CHANCE TO FIND OUT LATER (IF
YOU WANT TO KNOW) HOW MANY WORDS ARE ON MY LIST. @

EACH TIME YOU SEE THE *, TYPE IN ONE WORD WHICH YOU HAVE MADE FROM THE
LETTERS IN TOWN @

IF I THINK YOUR WORD IS A GOOD WORD, I WILL TELL YOU THE TOTAL NUMBER
OF WORDS YOU HAVE. @

IF I AM NOT SURE ABOUT YOUR WORD, I WILL LET YOU KNOW. @

NO PROPER NAMES, ABBREVIATIONS, CONTRACTIONS, OR FOREIGN WORDS ALLOWED.
@

IF AT ANY TIME YOU WANT TO STOP PLAYING, JUST TYPE -- STOP. @

READY?
TELL ME THE FIRST WORD YOU CAN MAKE FROM: TOWN
*TO
YOU NOW HAVE 1 WORD.
*NO
YOU NOW HAVE 2 WORDS.
*ON
YOU NOW HAVE 3 WORDS.
*OWN
YOU NOW HAVE 4 WORDS.
*WON
YOU NOW HAVE 5 WORDS.
*NOW
YOU NOW HAVE 6 WORDS.
*NOT
YOU NOW HAVE 7 WORDS.
I THOUGHT YOU WOULD LIKE TO KNOW THERE ARE ONLY THREE MORE WORDS ON MY
LIST.
*TON
YOU NOW HAVE 8 WORDS.
*TOW
YOU NOW HAVE 9 WORDS.
```

```
*WONT
THAT WORD IS NOT ON MY LIST, BUT I COULD BE WRONG. CHECK WITH YOUR
TEACHER WHEN YOU ARE THROUGH PLAYING. I WILL NOT COUNT IT FOR NOW.
 *TWO
YOU NOW HAVE 10 WORDS.
CONGRATULATIONS. YOU ARE A GOOD WORDHOUND!
IF YOU WANT TO PLAY AGAIN, TYPE -- WORDHUNT 1 WHEN YOU SEE THE ARROW.
←WORDHUNT 1
DO YOU KNOW HOW TO PLAY?
*YES
TELL ME THE FIRST WORD YOU CAN MAKE FROM: SAME
*ME
YOU NOW HAVE 1 WORD
*MA
YOU NOW HAVE 2 WORDS.
*AM
YOU NOW HAVE 3 WORDS.
I THOUGHT YOU WOULD LIKE TO KNOW THERE ARE ONLY THREE MORE WORDS ON MY
LIST.
 *AM
YOU HAVE ALREADY GIVEN ME THAT WORD.
*SEAM
YOU NOW HAVE 4 WORDS.
*AS
YOU NOW HAVE 5 WORDS.
*SEA
YOU NOW HAVE 6 WORDS.
CONGRATULATIONS. YOU ARE A GOOD WORDHOUND!
IF YOU WANT TO PLAY AGAIN, TYPE -- WORDHUNT 2 WHEN YOU SEE THE ARROW.
←WORDHUNT 2
DO YOU KNOW HOW TO PLAY?
*YES
TELL ME THE FIRST WORD YOU CAN MAKE FROM: DREAM
*DEAR

        :    :
```

```
-HANGMAN
I AM GOING TO PICK A MYSTERY WORD FROM MY WORD BANK.  YOU CAN TRY TO
GUESS MY WORD.
I WILL GIVE YOU A CLUE. I WILL PRINT A DASH FOR EACH LETTER IN THE
WORD.  IF YOU SEE ---, THE MYSTERY WORD HAS THREE LETTERS.
I WILL ASK YOU TO GUESS A LETTER. IF YOUR LETTER IS IN MY WORD I WILL
PUT IT WHERE IT BELONGS.
YOU HAVE SEVEN CHANCES TO GUESS MY WORD. THE FIRST TIME YOUR LETTER IS
NOT IN MY WORD I WILL START SPELLING HANGMAN. EACH TIME YOUR GUESS DOES
NOT WORK, I WILL ADD A LETTER. IF HANGMAN IS ALL SPELLED OUT, I WILL
TELL YOU MY WORD AND GIVE YOU A CHANCE TO TRY ANOTHER WORD.

READY? HERE WE GO!

---
WHAT IS YOUR GUESS?
*A
--- H
WHAT IS YOUR GUESS?
*O
-O- H
WHAT IS YOUR GUESS?
*D
-O- HA
WHAT IS YOUR GUESS?
*P
-O- HAN
WHAT IS YOUR GUESS?
*T
-O- HANG
WHAT IS YOUR GUESS?
*F
-O- HANGM
WHAT IS YOUR GUESS?
*W
-OW HANGM
WHAT IS YOUR GUESS?
*C
-OW HANGMA
WHAT IS YOUR GUESS?
*N
-OW HANGMAN
SORRY, YOUR MAN HAS BEEN HANGED.
YOUR WORD WAS HOW
DO YOU WANT TO PLAY AGAIN?
*YES
---
WHAT IS YOUR GUESS?
*U
--U
WHAT IS YOUR GUESS?
*O
-OU
WHAT IS YOUR GUESS?
*Y
YOU
CONGRATULATIONS! YOU'VE GOT IT!
```

```
----------------------------------------------------------------------
1                                                                    1
1                XX       XXXXXXX   XXXXXXX   XXXXXXX                1
1                XX       XXXXXXX   XXXXXXX   XXXXXXX                1
1                XX       XX   XX   XX   XX   XX   XX                1
1                XX       XX   XX   XX        XX   XX                1
1                XX       XX   XX   XX        XX   XX                1
1                XX       XX   XX   XX XXXX   XX   XX                1
1                XX       XX   XX   XX XXXX   XX   XX                1
1                XX       XX   XX   XX   XX   XX   XX                1
1                XXXXXX   XXXXXXX   XXXXXXX   XXXXXXX                1
1                XXXXXX   XXXXXXX   XXXXXXX   XXXXXXX                1
----------------------------------------------------------------------
        1111
        1111
        1111
        1111                           OOOOO
        1111                         OOOOOOOOOO
        1111                      /OOOOOOOOOOOOO
        1111                     "OOOOOOOOOOOOOOO
        1111                  OO"OOOOOOOOOOOOOOOOO
        1111   ------OOOOOOOOOOO██OOOOOOOOOOOOOOOOO
    -"""" ----OOOOOOOOOOOOOOOOOO██OOOOOOOOOOOOOOOOO
    ██████ OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
    ██████-OOOOOOOOOOOOOOOOOOOOOOOOOOOO/OO-OOOOOOOO
     █████"OOOOOOOOOOOOOOOOOOOOOOOOOOOO"████████████
       ███"OOOOOOOOOOOOOOOOOOOOOOOOOOO"██████████████
        11"OOOOOOOOOOOOOOOOOOOOOOOOOO"███████████████
        11"OOOOOOOOOOOOOOOOOOOOOOOOO"████████████████
        111"OOOOOOOOOOOOOOOOOOOOOOO"█████████████████
        1111"OOOOOOOOOOOOOOOOOOOOO"██████████████████
        1111 "OOOOOOOOOOOOOOOOOOO"██████████████████
        1111   """"""OOOOOOOOOOOO"█████████████████
        1111           "OOOOOOOO"██████████████████
        1111             "OOOOOO"████████████████
        1111             "OOOOOO"███████████████
        1111             "OOOO"  "█████████████
      -----             "OOOO"     ██████████
     /OOOOOO"          "OOOOO"      ██████
     /OOOOOOO"""     ""OOOOO"
     1OOOOOOOOOOO""OOOOOOOOO"
      ------"""OOOOOOOOOOOOO"
       1111    "OOOOOOOOOOOOO"
       1111   "OOOOOOO"OOOOOO""
       1111  "OOOOOOOO"OOO"OO"██
       ----  "OOOOOOOO"OOO"OO"███
            "OOOOOOOOO"OOO"OO"████
           "OOOOOOOOOO"OOO"OOO"█████
          "OOOOOOOOOOO"OOO"OOOO██████
          "OOOOOOOOOOO"OOO"OOOOO██████
           "OOOOOOOOO"OOOOOO"OOOOOOOOO"
            "OOOOOOO"OOOOOOOO"OOOOOOOO"
             "OOOOOO"OOOOOOOO"OOOOOOOO"
              "OOOOOO"-------"OOOOOOO"
                "OOOOOOOOOOOOOOOOOOO"
                  "OOOOOOOOOOOOOOOO"
                   "OOOOOOOOO"OOOOOOOO
                   "OOOO           OOOO O
         OOOO     OOO"              OOOOOOO
        OOOOOOOOOOOOO               OOOOOOOO
         OOOOOOOOOOO              OOOOOOOO
           OOOOOOOO               OOOOOOO
             OOOOOO              OOOOOO
              """""              """"
```

4. Junior High School Teaching Experiment

We started the LOGO experiment at Muzzey Junior High School in Lexington, Massachusetts, in September 1968. A class of twelve seventh-grade students, in the median range of mathematical performance, was selected at random (subject to the constraint - six boys and six girls) from a population of about two hundred students at the school at the same mathematics level. Mrs. Marjorie Bloom taught the first part of the course introducing the children to LOGO. Miss Cynthia Solomon and Dr. Seymour Papert taught the class from January 1969, continuing with a LOGO treatment of arithmetic and algebra.

4.1 Design and Operation of the Course

Course Design

We chose to develop a course in introductory algebra because most high school students have enormous difficulties with formal concepts and problem-solving in this subject. Moreover, much of the content of introductory algebra can be naturally treated in a thinly disguised form through teaching LOGO. The algebraic concepts of variable, literal, formula, equation, etc. have *conceptually* very clear analogs in LOGO. Thus, the concept of building LOGO things and referring to them by name is used to introduce the concept of *variable*. The LOGO concepts of conditional operation and predicate are used to introduce the mathematical idea of *equation*, and the LOGO concept of *procedure* is used to introduce the mathematical idea of *function*.

We gradually made the transition from the teaching of LOGO to traditional material by programming problems with a more numerical flavor and with a structure biased towards algebraic ideas (for example, by the construction of "search" programs to find objects - numerical or not - to satisfy given sets of conditions). After experience with these programs, students were introduced to algebraic procedures for solving equations. These gave rise to a new set of programming projects, to solve algebraic equations by symbolic methods.

Operation

Six computer terminals were installed in the classroom. The students' time was spent partly in classroom discussion, partly in designing programs, and partly in working with these programs at the computer terminal. They did most of their computer work individually but sometimes worked in pairs. The scheduling of time for students among these three classroom activities varied from day to day. Typically a student spent about half of the period in classroom discussion and half in writing and debugging programs, but there were occasional days, not liked by the students, in which they did not use the computer terminal at all.

The individual student work at the computer terminal was closely integrated with the teaching presentation. Some of the work at the terminals was relatively unstructured; some work assignments were very tightly specified. The practice varied among the teachers and, for a given teacher, across the various units taught.

4.2 LOGO Teaching Materials

The teaching objective of the initial weeks of the course was to impart fluency in the use of LOGO. A detailed outline of this part of the course is given next. The work was organized into three overall phases covering (1) the formal elements of LOGO programming, (2) debugging techniques and practice in their use, and (3) various projects to consolidate and apply the concepts treated.

Samples of the classroom presentations, laboratory assignments, and students' work are shown in subsequent pages.

Course Outline

Week	Topics
1	Computer Languages; Formal Instructions; Interactive Operation of a Computer; The LOGO Language; LOGO Things, Words, Sentences, Literals, Names; The PRINT Command; Operations on Things; Naming; The Operations FIRST, LAST, BUTFIRST, BUTLAST, COUNT.
2	The EMPTY Word; SUM and DIFF; Inputs and Outputs; Chaining of Operations; Order of Operations; Correcting Typing Errors, Backslash, Rubout; Bugs.
3	Standard Bugs, Giving Names to Bugs; Describing the action of a given instruction; Writing an instruction which has a given effect; Sequences of instructions, Procedures; LOGO Programs; Writing and Running of Programs.

Week	Topics
4	Listing of Programs; Program Editing; Writing programs which have prescribed effects; Checking programs with test inputs; Simulating the operation of a program.
5	The Conditional Operation and Tests; Simple Recursive Procedures; Recursive Programs with Tests.
6	Simulating the Operation of Recursive Programs; Recursively Defined Commands and Operations. The TRACE Command.
7	Chaining of Procedures; Embedding of Procedures; Debugging Problems.
8-9	Debugging Aids - Little Men Pictures, Round-Analysis; Storing and Retrieving of Programs - LOGO Filing Facilities.
10-11	Building Program Complexes; Projects; Interactive Programming - Game-playing and Quiz programs; Message Coding and Decoding programs.
12	Predicates; Special Names; Extension and Generalization of Programs; Standard form of Instructions in Recursive Procedures.

Formal Elements

LOGO was taught to the seventh-grade students in a graduated presentation sequence proceeding from the simplest elements, through instructions and expressions compounded from the elementary (built-in) LOGO operations, to complex and sometimes highly recursive procedures built as sequences of instructions. The first five weeks of the course were primarily devoted to teaching the formal structure of LOGO. Examples of the teaching materials, in the form of classroom assignments and laboratory work at the computer terminal, follow.

The first sample, Tips on Using the Teletype, was one of many handouts describing LOGO and its use. Six assignments follow this. The first three - CHALLENGE, NAMING THINGS, and A CODED MESSAGE FROM NANCY - are representative examples of numerous exercises given to the class in writing instructions and making and using names. The last of these assignments was: make up a message with LOGO instructions. Shown here is the response written by one of the students, Nancy. The students liked to exchange such "secret" LOGO messages.

Much of the elementary formal material used in the Muzzey classroom was similar to that used in the elementary school work at Emerson (described in the previous section of this report), though the form of the presentation was different. (The elementary school presentations was largely in the form of programmed lessons.) During the first part of their course, however, the seventh-grade students had more work with procedures, including somewhat more difficult procedures, than did the elementary school children. The last three assignments included here concern work with writing procedures.

The first of these, SOME INCOMPLETE PROCEDURES, is an exercise in completing the definition of some functions. Included is a "silly" function whose output is independent of its input. The assignment on SOME TESTING PROCEDURES is somewhat more open-ended. The student is to write procedures to perform various specified tests. He is then to write several programs which use such testing procedures.

The last assignment, TWO PROCEDURES TO COMPLETE, requires the student to incorporate numerical test operations, such as GREATER, in interactive procedures.

Some original student work along these lines is reproduced following these assignments.

Tips on Using the Teletype

In many ways the teletype works like an ordinary electric typewriter. However, there are some differences which you will need to know.

1. The teletype types only in capital letters. You will need to use the SHIFT key to obtain such symbols as (,), #, ", etc.

2. If you make a typing error, do not worry! There are several ways to correct errors.
   a. To erase a single letter at a time -- press the backslash key (\\).
   b. To erase a word at a time -- press the CONTROL key and the letter W.
   c. To erase a line -- press RUBOUT key.

3. If you are typing in something that is too long for one line, press LINEFEED and continue typing after the carriage has returned.

4. When you press the RETURN key you signal to the computer that you have completed the line of typing.

5. There is no back space on a teletype. If you have put in a space where you do not want a space, you must erase the space in the same way you would erase a mistyped letter -- by pressing the BACKSLASH key (\\).

CHALLENGE!!!!

Can you make up LOGO instructions which will obtain the responses shown using the words given?

	Word	Computer Response
EXAMPLE:	"LIGHT"	H

One instruction might be: PRINT LAST OF BUTLAST OF "LIGHT"

	Word You Are to Use	Desired Computer Response
1.	"FRIEND"	R

PRINT FIRST OF BUTFIRST OF "FRIEND"

2.	"DEAR"	EA

PRINT BUTFIRST OF BUTLAST OF "DEAR"

PRINT WORD OF BUTFIRST OF "DEAR" AND LAST OF BUTLAST OF "DEAR"

3.	"A"	

PRINT BUTLAST OF "A"

PRINT BUTFIRST OF "A"

4.	"BAR"	BARB

PRINT WORD OF "BAR" AND FIRST OF "BAR"

5.	"MY"	YM

PRINT WORD OF LAST OF "MY" AND FIRST OF "MY"

PRINT WORD OF BUTFIRST OF "MY" AND FIRST OF "MY"

NAMING THINGS

Type the following CALL instructions to the computer. How do you think the computer should answer if you type in the following PRINT instructions. Make all your responses first -- then check with the computer.

```
CALL
   THING:   "LUCY"
   NAME:    "DOCTOR"

CALL
   THING:   "PATTY"
   NAME:    "ROPE JUMPER"

CALL
   THING:   "SILLY"
   NAME:    "LUCY"
```

Read carefully before making your responses!

Instruction	Your Response	Computer Response
PRINT "DOCTOR"	DOCTOR	
PRINT THING OF "DOCTOR"	LUCY	
PRINT THING OF THING OF "DOCTOR"	SILLY	
PRINT "ROPE JUMPER"	ROPE JUMPER	
PRINT THING OF "ROPE JUMPER"	PATTY	
PRINT THING OF THING OF "ROPE JUMPER"	(THE EMPTY WORD)	
PRINT THING OF "LUCY"	SILLY	
PRINT THING OF THING OF "LUCY"	(THE EMPTY WORD)	

A CODED MESSAGE FROM NANCY

Some names to give the computer:

CALL
THING: "HE"
NAME: "SHE"

CALL
THING: "TYPE"
NAME: "PRESS"

CALL
THING: "EVERYTHING"
NAME: "NOTHING"

CALL
THING: "TELL"
NAME: "APPLE"

CALL
THING: "YOU"
NAME: "ME"

CALL
THING: "IT"
NAME: "HER"

CALL
THING: "ASK"
NAME: "TWO"

T, HE
1,2

TEL, E, TYPE
3,4,5

TELL, S
6,7

YOU
8

EVERYTHING
9

YOU
10

ASK
11

IT
12

INSTRUCTIONS:

1. PRINT "T" T
2. PRINT /SHE/ HE
3. PRINT BUTLAST OF /APPLE/ TEL
4. PRINT "E" E
5. PRINT /PRESS/ TYPE
6. PRINT /APPLE/ TELL
7. PRINT "S" S
8. PRINT /ME/ YOU
9. PRINT /NOTHING/ EVERYTHING
10. PRINT /ME/ YOU
11. PRINT /TWO/ ASK
12. PRINT /HER/ IT

SOME INCOMPLETE PROCEDURES

Here are some procedures for you to look at. Below each procedure are some examples of what the procedure does. However, some of the instructions are missing. Your job is to study what the procedure does and see if you can fill in the missing instructions.

(1)

```
 TO ARG /X/
 10 PRINT FIRST OF /X/
 15 ____________________________
 END
ARG DEFINED

 ARG "APPLE"
A
E

 ARG "HELLO"
H
O

 ARG "T"
T
T
```

(2)

```
 TO TOP /X/
 10 ____________________________
 END
TOP DEFINED

 PRINT TOP "TELL ME A STORY"
ME, TOO

 PRINT TOP "I LIKE CANDY"
ME, TOO

 PRINT TOP "THROW HIM OUT"
ME, TOO
```

(3)

```
 TO JOB /X/
 10 PRINT FIRST OF /X/
 15 PRINT LAST OF /X/
 20 ______________________________
 END
JOB DEFINED

 PRINT JOB 26
2
6
8

 PRINT JOB 489
4
9
13

 PRINT JOB 3
3
3
6
```

(4)

```
 TO HOE /X/
 10 PRINT FIRST /X/
 20 ______________________________
 30 RETURN HOE /X/
 END
HOE DEFINED
 PRINT HOE "GREEN"
G
REEN
G
REEN
G
REEN
G
I WAS AT LINE 20 IN HOE
 PRINT HOE 100000
1
00000
1
00000
1
I WAS AT LINE 20 IN HOE
```

SOME TESTING PROCEDURES

Write programs for the following:

1. Procedure to compare two words to see if they are the same.
2. Procedure to compare two words to see if they have same count.
3. Procedure to find out if two words have same first letter.
4. Procedure to find out if word begins with double letter.
5. Procedure to find out if word has same first and last letters.
6. Procedure to find out if a word begins with a "b" or a "c".
7. Procedure to find out if word begins with "bl".
8. Procedure to find out if a number is even or odd.

Using a testing procedure to find out if another procedure should be used.

1. If word ends in "ed", remove "ed".
2. If word begins with bl, change bl to sl.
3. If number is even, double it; if number is odd, subtract 1 from it.
4. If letter is vowel, print "TRUE"; otherwise, print "FALSE".
5. If word begins with a vowel, print it; otherwise print "BAD WORD".

TWO PROCEDURES TO COMPLETE

1. OBJECT OF PROCEDURE:

A. This procedure requests that the player type in a number.
B. It then requests that the player type in a different number.
C. It compares the two numbers typed in to see which is larger.
D. If the first number is larger than the second number, it returns "THE FIRST NUMBER IS LARGER THAN THE SECOND NUMBER".
E. Otherwise, it returns "THE SECOND NUMBER IS LARGER THAN THE FIRST NUMBER".

Here is the beginning of the procedure. You complete and test it.

```
TO COMPARE
10 PRINT "TYPE IN A NUMBER WHEN YOU SEE * "
20 REQUEST "FIRSTNUMBER"
30 PRINT "TYPE IN A DIFFERENT NUMBER NOW."
40 REQUEST "SECONDNUMBER"
      .
      .                 You complete the rest of the procedure.
      .
      .
      .
      .
```

2. OBJECT OF PROCEDURE:

A. This procedure requests that the player type in a number.

B. It then tests to find out if the number is between 100 and 1000.

C. If the number is less than or equal to 100, it returns "YOUR NUMBER IS LESS THAN OR EQUAL TO 100."

D. If the number is greater than or equal to 1000, it will return "YOUR NUMBER IS GREATER THAN OR EQUAL TO 1000."

E. If the number is between 100 and 1000, it will return "YOUR NUMBER IS BETWEEN 100 AND 1000."

HINT: You will need to write more than one test question in this procedure.

```
TO NUMTEST

10 PRINT "TYPE IN ANY NUMBER WHEN YOU SEE * "

20  REQUEST "NUMBER"

      .
      .
      .                    You complete the procedure.
      .
      .
      .
```

Illustrations of Student Work

The following typescripts show a student's work writing three procedures. The first one, GAME, is a test procedure which returns "YES" if its input is a 3-letter word (or a 3-word sentence), and "NO" otherwise.

The second one, SPLITTHREE, types the first, second, and last letters of its input (if the input is a word, or the first, second, and last words if its input is a sentence) vertically down the page. This procedure uses another procedure, SECOND, as part of its definition.

The last of the three procedures, THREE, uses the other two. It performs GAME to test whether or not to perform SPLITTHREE.

```
←TO GAME /Y/
>10 IS /Y/ 3 LETTER WORD     (A typical early bug.  The inten-
                             tion is clear but it was not
                             expressed as a legal LOGO
                             instruction.)
>10 IS COUNT /Y/ 3           (It was immediately fixed)
>20 IF YES RETURN "YES"
>30 IF NO RETURN "NO"
>END
GAME DEFINED

←PRINT GAME OF "CAT"
YES

←PRINT GAME OF "LONGWORD"
NO

←P GAME OF "ELEPP"
NO
```

```
←P GAME OF "THREE WORD SENTENCE"    (GAME is tested with sen-
YES                                 tences as well as words)
←P GAME OF FIRST OF "THE GRASS IS GFRREEN"  (And with outputs
YES                                         of operations on
←P GAME OF "MOI"                            sentences)
YES
←P GAME OF "123"                  (It still works with words)
YES



TO SPLITTHREE /A/
12 PRINT FIRST OF /A/
13 PRINT SECOND OF /A/    (He incorrectly assumed that there
14 PRINT LAST OF /A/      was a built-in LOGO procedure,
END                       SECOND)

←SPLITTHREE OF "TWO"      (SPLITTHREE started out all right;
T                         it printed the FIRST OF "TWO";
SECOND NEEDS A MEANING.   then it ran into a problem, and
I WAS AT LINE 13 IN SPLITTHREE                       stopped.)
←

TO SECOND /A/                          (The student then wrote
12 RETURN FIRST OF BUTFIRST OF /A/     a SECOND procedure)
END

←SPLITTHREE "TWO"                 (Now SPLITTHREE works)
T
W
O
←SPLITTHREE "THREE"
T
H
E
←SPLITTHREE "ABRA CADABRA"
ABRA
CADABRA
CADABRA
←

←TO THREE /WORD/
>10 IS GAME OF /WORD/ "YES"
>20 IF NO PRINT "NO"
>30 IF YES SPLITTHREE /WORD/
>END
THREE DEFINED
```

```
←THREE "123"
1
2
3
←THREE "ONE TWO THREE"
ONE
TWO
THREE
←THREE "ONE"
O
N
E
←THREE "THREE"
NO
←
```

In responding to the second part of the assignment TWO PROCEDURES TO COMPLETE, the teacher expected that students' procedures would use the GREATER operation. (The output of GREATER OF /M/ AND /N/ is /M/ if /M/ ≥ N; else it is /N/.) The students did not like this notation for GREATER because they were used to the idiom /M/ is GREATER THAN /N/. (At the same time, in a classroom vote, they overwhelmingly rejected the option of replacing GREATER OF /M/ AND /N/ with MAXIMUM OF /M/ AND /N/, though the latter did not conflict with familiar usage. Possibly this was because they didn't like <u>any</u> changes in the language. They were used to the existing instructions and preferred these to new and uncertain ones, even if the new ones appeared to be better.)

Two students found a way of getting around this problem. Note how they avoided the use of GREATER in the following procedure.

```
TO NUMTEST
10 PRINT "TYPE IN A NUMBER WHEN YOU SEE *"
20 REQUEST /NUM/
30 IS COUNT OF /NUM/ 1
40 IF YES RETURN "YOUR NUMBER IS LESS THAN 10"
50 IS COUNT OF /NUM/ 2
60 IF YES RETURN "YOUR NUMBER IS LESS THAN 100"
70 IS COUNT OF /NUM/ 3
80 IF YES RETURN "YOUR NUMBER IS LESS THAN 1000"
90 RETURN "YOUR NUMBER IS 1000 OR LARGER"
END
```

Independence and originality were always encouraged. After the students were praised for their clever idea, they were asked to write an alternate procedure, this time using GREATER. Their first effort, called NUM, was as follows.

```
TO NUM
10 PRINT "TYPE IN A NUMBER WHEN YOU SEE*"
20 REQUEST "N"
30 IS /N/ GREATER "1000"
40 IF YES RETURN "YOUR NUMBER IS 1000 OR LARGER"
50 IF NO RETURN "YOUR NUMBER IS LESS THAN 1000"
END
```

It has the standard GREATER bug in line 30. After a major debugging episode, line 30 was changed to:

```
30 IS /N/ GREATER OF /N/ AND "1000"
```

Now the NUM procedure worked. The students retained NUMTEST which they obviously preferred in their files.

There was a great deal of similar work of this kind during the first weeks of the course. The early work was designed primarily to help students become proficient in the elements of LOGO programming. As the students attained a modicum of fluency in LOGO, they were introduced to constructive problems of somewhat richer structure requiring more problem-solving "know-how".

Heuristic Work

Making the formal elements of LOGO programming accessible does not guarantee that students will be able to write their own programs to solve problems, or even to understand relatively simple and transparent programs representing solutions. Along with teaching programming as a formal language, we need to teach students how to simulate the operation of a program to understand what it does, to decide if a program "works", and to find errors or "bugs" in programs that do not work.

A large part of the course was spent on working with programs that did not quite do what they were supposed to, and trying to fix (or "debug") them. This work has very direct relevance to the teaching of mathematics, not merely programming. The process of "debugging" programs gives students a rich base of personal experiences with the activity of solving mathematical problems. It enables them to confront and better understand their own thought processes. Thus, it is a valuable means of contributing to teaching the informal, intuitive, heuristic aspects of mathematical thinking and work.

A first step in teaching students debugging is teaching them to model and simulate the operation of a program. Traditionally, flow diagrams are used to present an overall model of a program. Since these do not give a good picture of the operation of recursive programs, we developed a new kind of diagram - "little men" pictures - to help students get a clear picture of how LOGO programs work. These are discussed next, in conjunction with the children's classroom and laboratory assignments. To illustrate the use of these diagrams, consider the procedure REVERSE.

REVERSE is a recursive operation whose output is the reverse of its input (i.e., its input written backwards). Thus, REVERSE OF "CAT" is "TAC".

```
TO REVERSE /WORD/
10 TEST IS /WORD/ /EMPTY/
20 IF TRUE OUTPUT /WORD/
30 OUTPUT WORD OF LAST /WORD/ AND REVERSE OF BUTLAST /WORD/
END
```

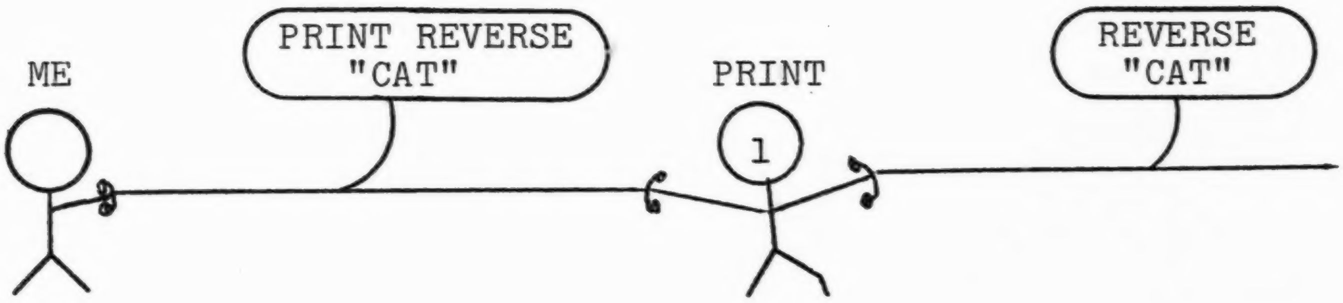
The procedure as written is too elegant - by being too compact its structure is hidden. We can make it more transparent by paraphrasing it into a form whose parts are more visible, functionally separable, and so nameable. For example,

```
TO REVERSE /WORD/
10 TEST IS /WORD/ /EMPTY/                        \     Check
20 IF TRUE OUTPUT /EMPTY/                        /  (Stop Rule)
30 MAKE
     NAME:  "NEWWORD"                            \  Preparation
     THING:  BUTLAST OF /WORD/                   |
40 MAKE                                          }
     NAME:  "LETTER"                             |
     THING:  LAST OF /WORD/                      /
50 OUTPUT WORD OF /LETTER/ AND REVERSE OF /NEWWORD/  Action
END
```

We now illustrate the use of "Little Men" diagrams in modeling the operation of this procedure.

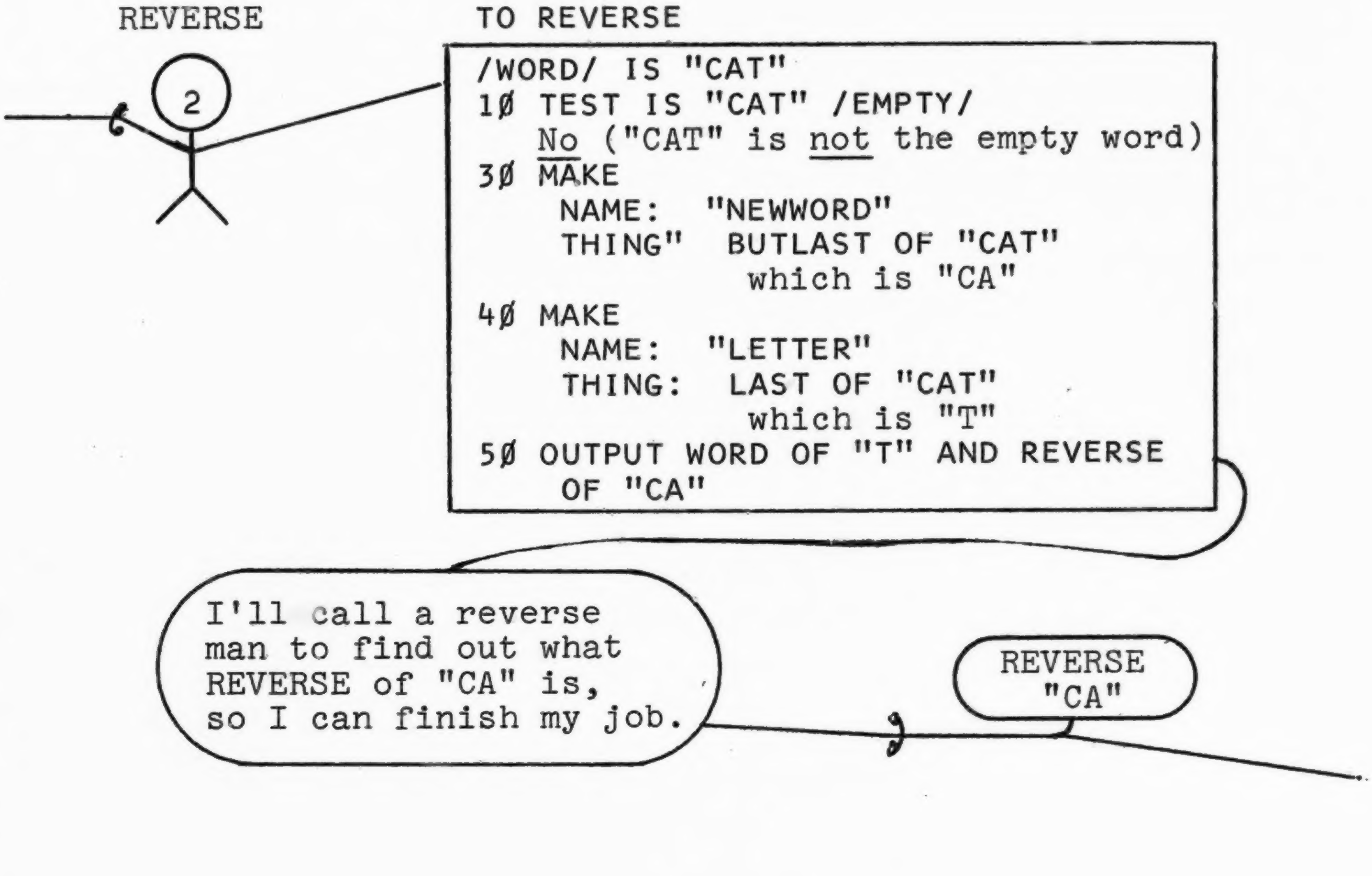
Little Men Pictures

Suppose I give the instruction PRINT REVERSE OF "CAT". In the first frame of the picture you see me calling a PRINT man and telling him what to print:

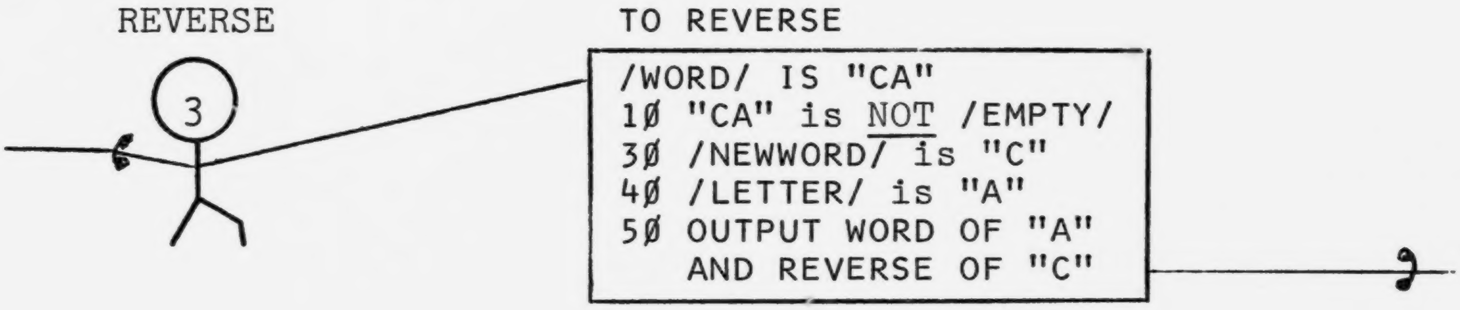

ME
PRINT REVERSE "CAT"
PRINT
1
REVERSE "CAT"

The PRINT man (he is labeled number 1) grabs the phone as soon as he sees the word REVERSE, and asks the REVERSE man (his number is 2) for help.

The REVERSE man carries out his procedure to the point where he needs to ask for help from another REVERSE man.


REVERSE
2
TO REVERSE
I'll call a reverse man to find out what REVERSE of "CA" is, so I can finish my job.
REVERSE "CA"

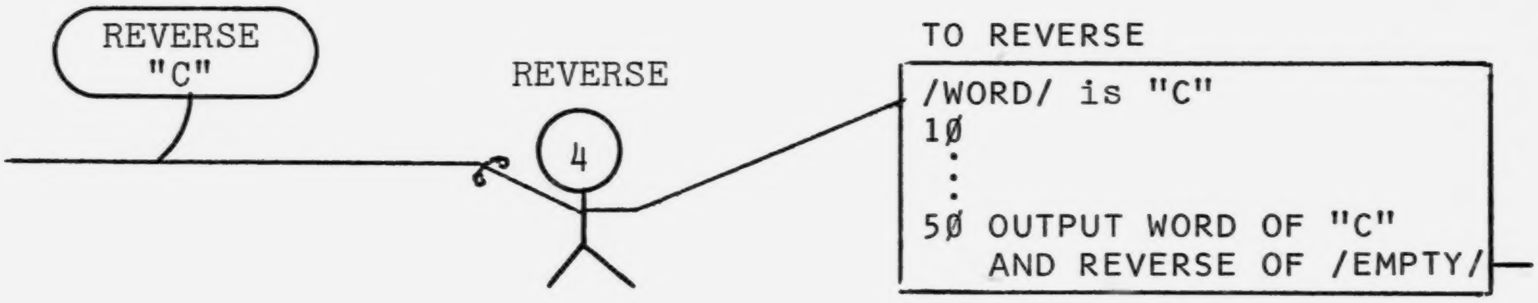
```
/WORD/ IS "CAT"
10 TEST IS "CAT" /EMPTY/
   No ("CAT" is not the empty word)
30 MAKE
     NAME:  "NEWWORD"
     THING"  BUTLAST OF "CAT"
               which is "CA"
40 MAKE
     NAME:  "LETTER"
     THING:  LAST OF "CAT"
               which is "T"
50 OUTPUT WORD OF "T" AND REVERSE
     OF "CA"
```


REVERSE
3
TO REVERSE

```
/WORD/ IS "CA"
10 "CA" is NOT /EMPTY/
30 /NEWWORD/ is "C"
40 /LETTER/ is "A"
50 OUTPUT WORD OF "A"
   AND REVERSE OF "C"
```

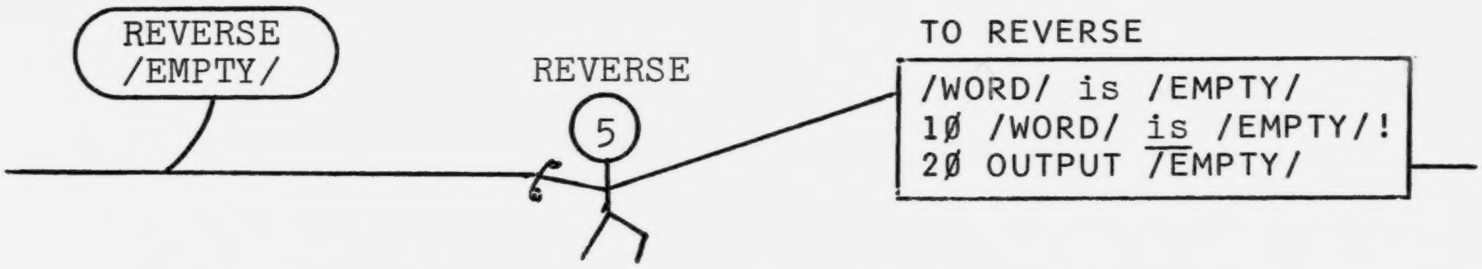
This REVERSE man (number 3) reads his procedure and carries on O.K. until he sees his line 50.

He needs a new REVERSE man to find REVERSE OF "C".


REVERSE
"C"
REVERSE
4
TO REVERSE

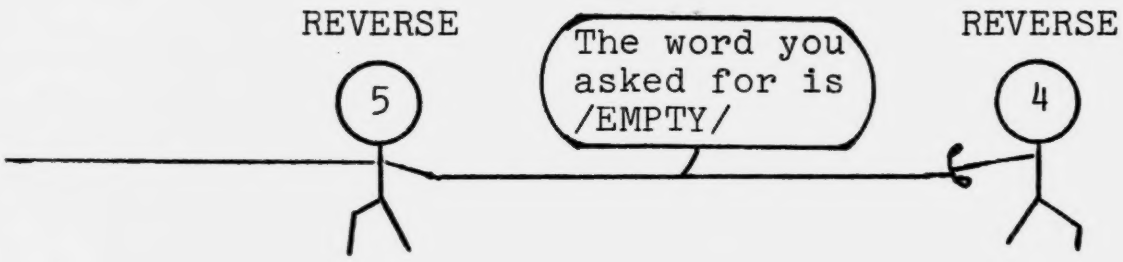
```
/WORD/ is "C"
10
 :
 :
50 OUTPUT WORD OF "C"
   AND REVERSE OF /EMPTY/
```

This new REVERSE man sees in his line 50 that he needs a REVERSE man to help him finish his job by finding REVERSE OF /EMPTY/.

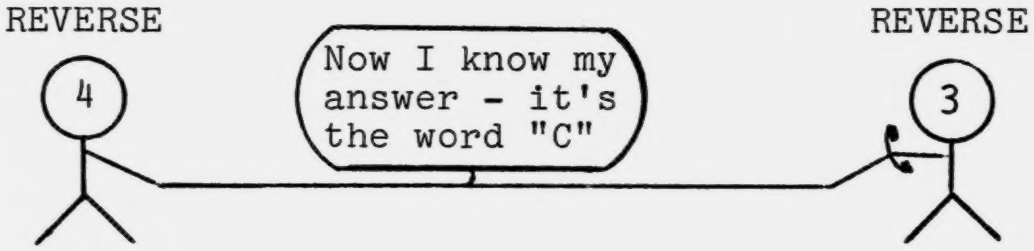

REVERSE
/EMPTY/
REVERSE
5
TO REVERSE

```
/WORD/ is /EMPTY/
10 /WORD/ is /EMPTY/!
20 OUTPUT /EMPTY/
```

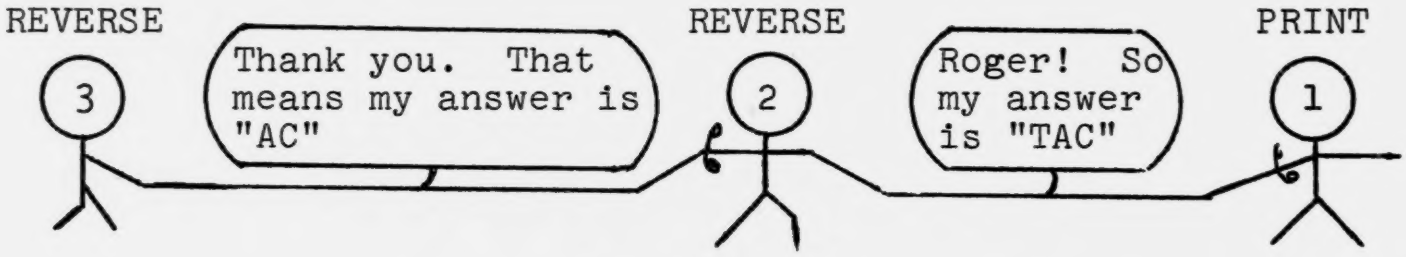
This REVERSE man (number 5) finishes his job and outputs his answer, which is /EMPTY/, to the man that called him.

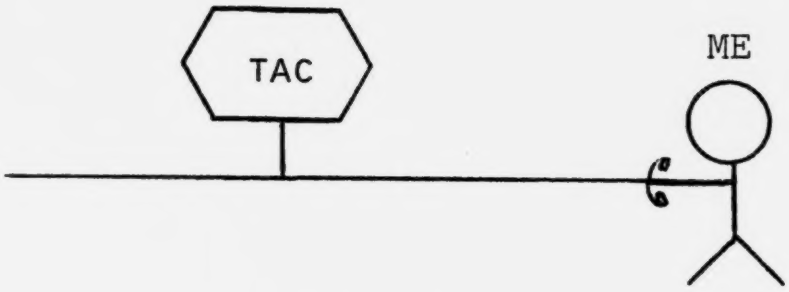

REVERSE
5
The word you asked for is /EMPTY/
REVERSE
4

Now, REVERSE man 4 can finish his line 50 and give his answer to the man who called him, REVERSE man 3.


REVERSE
4
Now I know my answer - it's the word "C"
REVERSE
3

Similarly:


REVERSE
3
Thank you. That means my answer is "AC"
REVERSE
2
Roger! So my answer is "TAC"
PRINT
1



The PRINT man prints the answer TAC for me. Voila! I hang up.

Exercises on Little Men Pictures

In these exercises we shall practice using the little-men-and-telephone idea to explain how a procedure works. As an example we use a procedure called TO XJOIN.

Purpose of TO XJOIN

This is a building procedure that extends a word by putting on extra X's in front of it.

Example: "CAT" is extended to "XXXXXCAT"

Inputs of TO XJOIN

We have to tell the procedure one thing: the original word (so it knows where to start). So the title will be

```
TO XJOIN /WORD/
```

Examples:

```
PRINT XJOIN "ANXIBAR"
XANXIBAR

PRINT XJOIN "NOMOROOM"
NOMOROOM

PRINT XJOIN "I"
XXXXXXXI
```

The Procedure

```
TO XJOIN /WORD/
10 IS COUNT OF /WORD/ 8
   (If it is then /WORD/ is the right length and
    can be returned)
20 IF YES RETURN /WORD/
   (If no, then we will extend /WORD/ by adding "X"
    in front of it)
30 CALL
    THING:  WORD OF "X" AND /WORD/
    NAME:  "NEWWORD"
  (Now we try the same thing again with /NEWWORD/)
40 RETURN XJOIN OF /NEWWORD/
END
```

Further Exercises

Consider another procedure called to EXPAND. This is a building procedure like XJOIN, but it has three inputs:

(1) The original word, so it knows where to start.
(2) The letter to be put on the front of /WORD/. (In XJOIN this was always "X".)
(3) The final length desired, so it knows when to stop. (In XJOIN this was always 8.)

So the title will be

```
TO EXPAND /WORD/ /LETTER/ /LENGTH/
```

Examples:

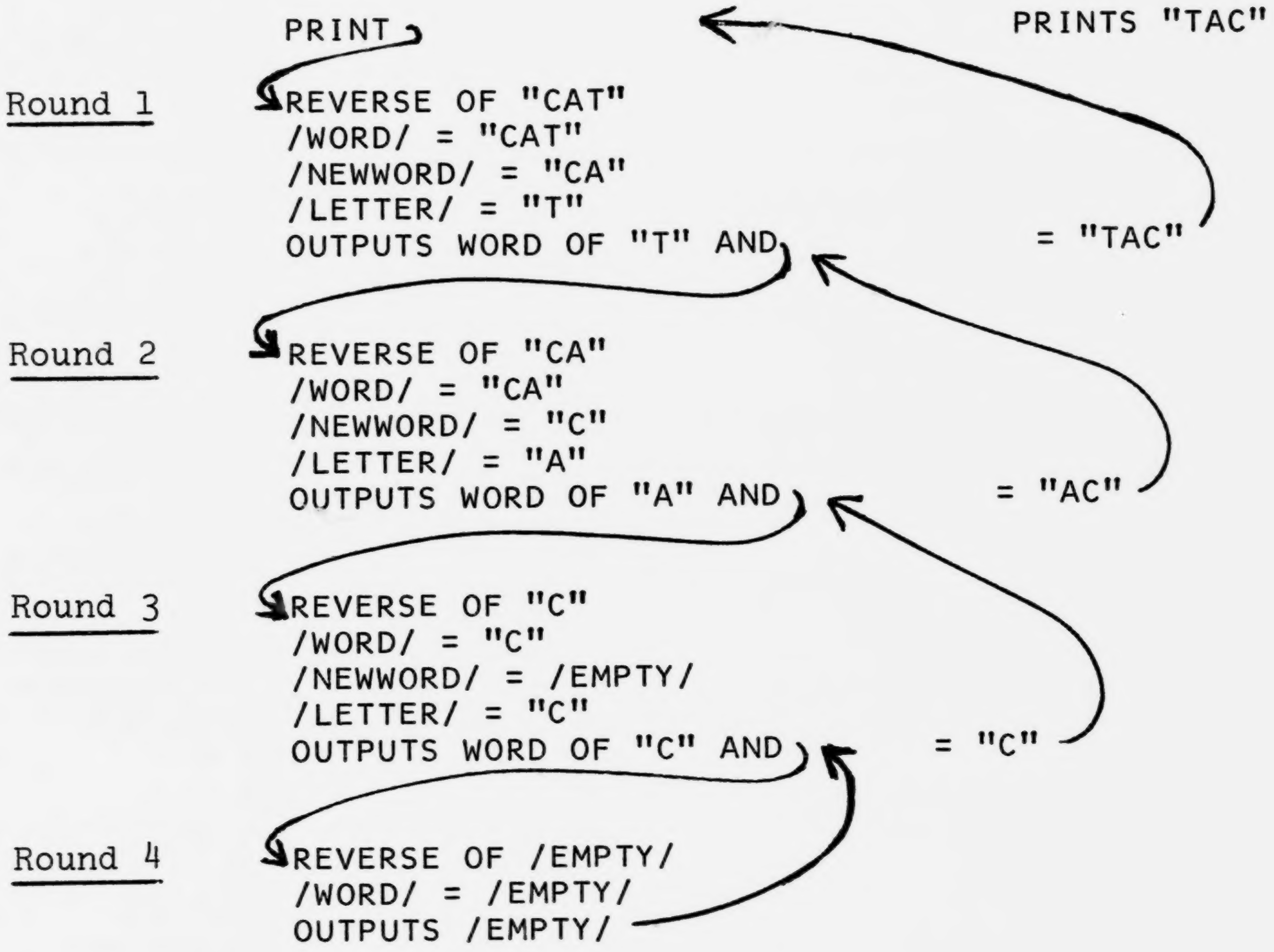
```
PRINT EXPAND "BOX" AND "A" AND 4
ABOX

PRINT EXPAND "S" AND "X" AND 10
XXXXXXXXXS
```

Write the procedure TO EXPAND. Then make a little men picture to show how the procedure works, given the instruction

```
PRINT EXPAND OF "DOG" "X" 5.
```

Round-Analysis

The analysis by "little men" becomes tedious and distracting once the principle is understood. Round-analysis, illustrated here for the same REVERSE procedure, is essentially a compact version of the "little men" diagram expressed in a form that is faster to write out.



```
               PRINT                                    PRINTS "TAC"

Round 1        REVERSE OF "CAT"
               /WORD/ = "CAT"
               /NEWWORD/ = "CA"
               /LETTER/ = "T"
               OUTPUTS WORD OF "T" AND          = "TAC"

Round 2        REVERSE OF "CA"
               /WORD/ = "CA"
               /NEWWORD/ = "C"
               /LETTER/ = "A"
               OUTPUTS WORD OF "A" AND        = "AC"

Round 3        REVERSE OF "C"
               /WORD/ = "C"
               /NEWWORD/ = /EMPTY/
               /LETTER/ = "C"
               OUTPUTS WORD OF "C" AND      = "C"

Round 4        REVERSE OF /EMPTY/
               /WORD/ = /EMPTY/
               OUTPUTS /EMPTY/
```

The TRACE Command

The TRACE command in LOGO allows the student to see the sequence of inputs and outputs as these develop in the course of running a program. Thus, TRACE automatically performs one of the main functions of round-analysis. The use of TRACE on the procedure REVERSE with the input "CAT" is shown in the following printout.

```
TO REVERSE /WORD/
10 TEST IS /WORD/ /EMPTY/
20 IF TRUE OUTPUT /EMPTY/
30 MAKE
    NAME:  "NEWWORD"
    THING:  BUTLAST OF /WORD/
40 MAKE
    NAME:  "LETTER"
    THING:  LAST OF /WORD/
50 OUTPUT WORD OF /LETTER/ AND REVERSE OF /NEWWORD/
END

←TRACE REVERSE
←PRINT REVERSE OF "CAT"

REVERSE OF "CAT"
  REVERSE OF "CA"
    REVERSE OF "C"
      REVERSE OF ""
      REVERSE OUTPUTS ""
    REVERSE OUTPUTS "C"
  REVERSE OUTPUTS "AC"
REVERSE OUTPUTS "TAC"
TAC
←
```

The procedure TO REVERSE is listed first. The instruction TRACE REVERSE is then executed. This informs LOGO that REVERSE is to be traced when it is used subsequently (until the TRACE is erased). When the instruction PRINT REVERSE OF "CAT" is executed, the successive calls of REVERSE are listed as they are made.

First the program calls for REVERSE OF "CAT", then for REVERSE OF "CA", "C", and "" (the empty word). As the output corresponding to each call is made and passed back, it is listed by TRACE. Each output is listed on a line having the same indentation as the corresponding call. Finally, the first "REVERSE man", REVERSE OF "CAT", can make its output "TAC" and pass it back to PRINT, which prints it.

The assignments that follow illustrate some of the debugging problems given to the class.

HELP!!!

My procedure CT doesn't work quite as well as it should.

THIS IS WHAT I WANTED IT TO DO:

```
←PRINT CT "HARD"
THIS WORD HAS AT LEAST 3 LETTERS.
←PRINT CT "GO"
THIS WORD HAS 1 OR 2 LETTERS.
←PRINT CT "DOG"
THIS WORD HAS AT LEAST 3 LETTERS.
←PRINT CT "I"
THIS WORD HAS 1 OR 2 LETTERS.
←
```

THIS IS THE PROCEDURE I WROTE:

```
←TO CT /WORD/
>10 IS COUNT /WORD/ "1"
>20 IS COUNT /WORD/ "2"
>30 IF NO RETURN "THIS WORD HAS AT LEAST 3 LETTERS."
>40 IF YES RETURN "THIS WORD HAS 1 OR 2 LETTERS."
>END
CT DEFINED
←
```

THIS IS WHAT ACTUALLY HAPPENED WHEN I USED MY PROCEDURE:

```
←PRINT CT "HARD"
THIS WORD HAS AT LEAST 3 LETTERS.
←PRINT CT "GO"
THIS WORD HAS 1 OR 2 LETTERS.
←PRINT CT "DOG"
THIS WORD HAS AT LEAST 3 LETTERS.
←PRINT CT "I"
THIS WORD HAS AT LEAST 3 LETTERS.
←
```

THE LAST ANSWER IS WRONG.

PLEASE HELP ME FIND THE BUGS!

MORE HELP!!!

This procedure leaves some funny spaces at the end.

THIS IS WHAT I WANTED IT TO DO:

```
←DRAW "XXXXXXX"
XXXXXXX
XXXXXXX
XXXXXXX
XXXXX
XXXXX
XXXXX
XXX
XXX
XXX
X
X
X
←
```

THIS IS THE PROCEDURE I WROTE:

```
←TO DRAW /X/
>10 PRINT /X/
>20 PRINT /X/
>30 PRINT /X/
>40 IS COUNT OF /X/ 0
>50 IF YES RETURN ""
>60 RETURN DRAW OF BUTFIRST OF BUTLAST /X/
>END
DRAW DEFINED
←
```

THIS IS WHAT HAPPENED WHEN I USED IT:

```
←DRAW "XXXXXXX"
XXXXXXX
XXXXXXX
XXXXXXX
XXXXX
XXXXX
XXXXX
XXX
XXX
XXX
X
X
X
```

(Can you get rid of all these spaces for me?)

```
←
```

STILL MORE HELP!!!

This procedure COUNTLESS is giving me trouble.

THIS IS WHAT I WANTED IT TO DO:

```
←PRINT COUNTLESS "ABCDEF"
ABCDEF
6
BCDEF
5
CDEF
4
DEF
3
EF
2
F
1
←
```

THIS IS THE PROCEDURE I WROTE:

```
←TO COUNTLESS /WORD/
>10 IS /WORD/ ""
>20 IF YES RETURN ""
>30 PRINT /WORD/
>40 PRINT COUNT OF /WORD/
>50 COUNTLESS /WORD/
>END
COUNTLESS DEFINED
←
```

THIS IS WHAT ACTUALLY HAPPENED WHEN I USED MY PROCEDURE:

```
←PRINT COUNTLESS "ABCDEF"
ABCDEF
6
ABCDEF
6
ABCDEF
6
ABCDEF
6
I WAS AT LINE 40 IN COUNTLESS.
←
```

PLEASE HELP!

An example of a standard situation where in the process of fixing one bug another is created, is shown in the following student's work in writing his own COUNTLESS. (He chose to start with his own program instead of editing the teacher's program shown on the previous page.)

```
TO COUNTLESS /WORD/
10 PRINT /WORD/
20 PRINT COUNT OF /WORD/
30 COUNTLESS BUTFIRST OF /WORD/
END
```

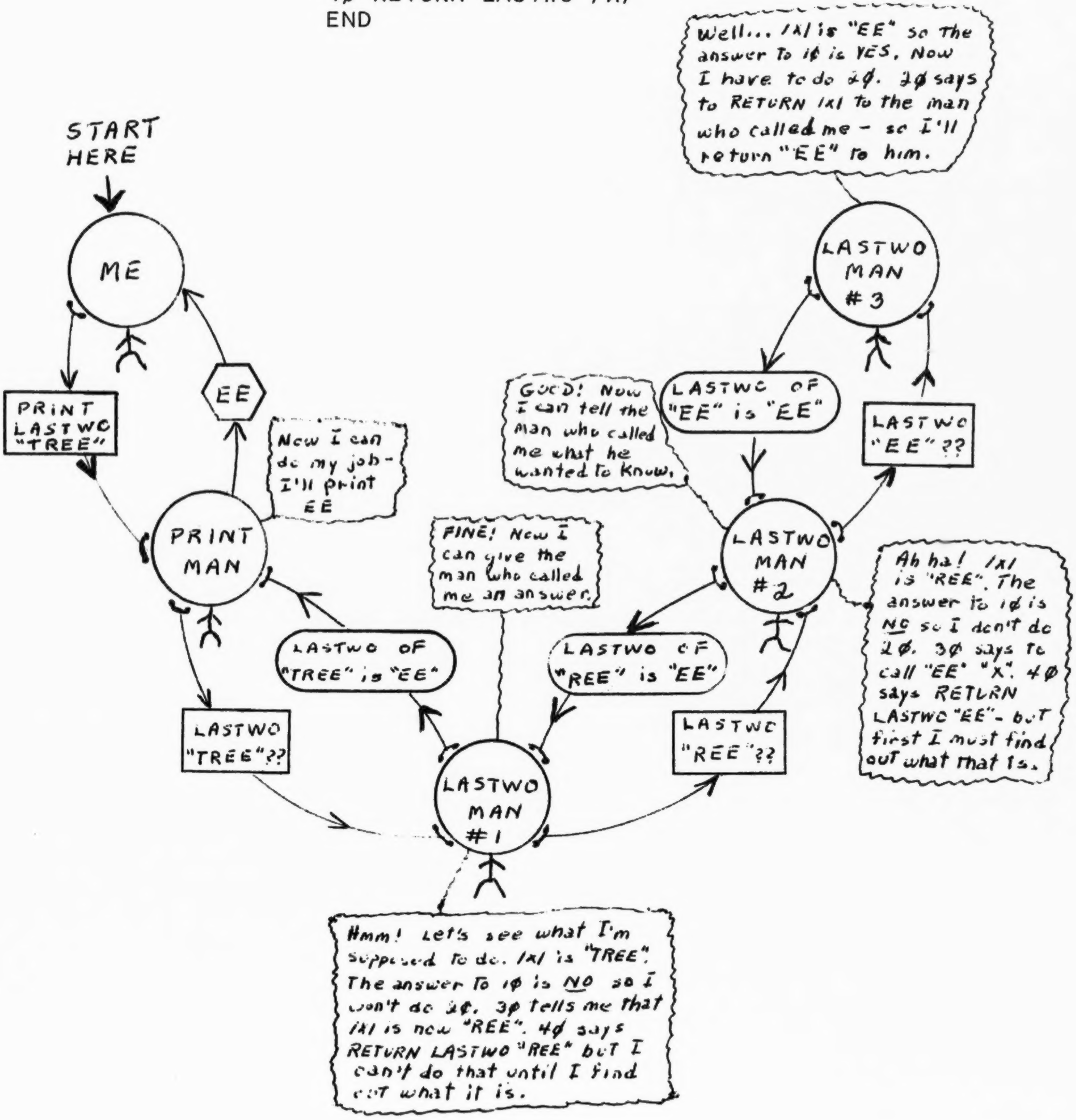
(The last line fixed the bug in the assigned program.)

```
←COUNTLESS "ABC"
ABC
3
BC
2
C
1
          (But it didn't stop)
0
          (Alternate printouts of the empty word
0          and its count, 0.)

0
.         (After several lines of repetition,
:          the break key was hit.)
I WAS AT LINE 20 IN COUNTLESS
     (The student had forgotten to include a stopping rule.
      He realized this and repaired his program.)
←EDIT COUNTLESS
>25 IS COUNT OF /WORD/ "1"
>27 IF YES RETURN /EMPTY/
>END
COUNTLESS DEFINED
←COUNTLESS "ABC"
ABC
3
BC
2
C
1
←
```

The next assignment, shown on the following two pages, is an exercise in Little Men diagramming. (The XJOIN procedure was discussed above.)

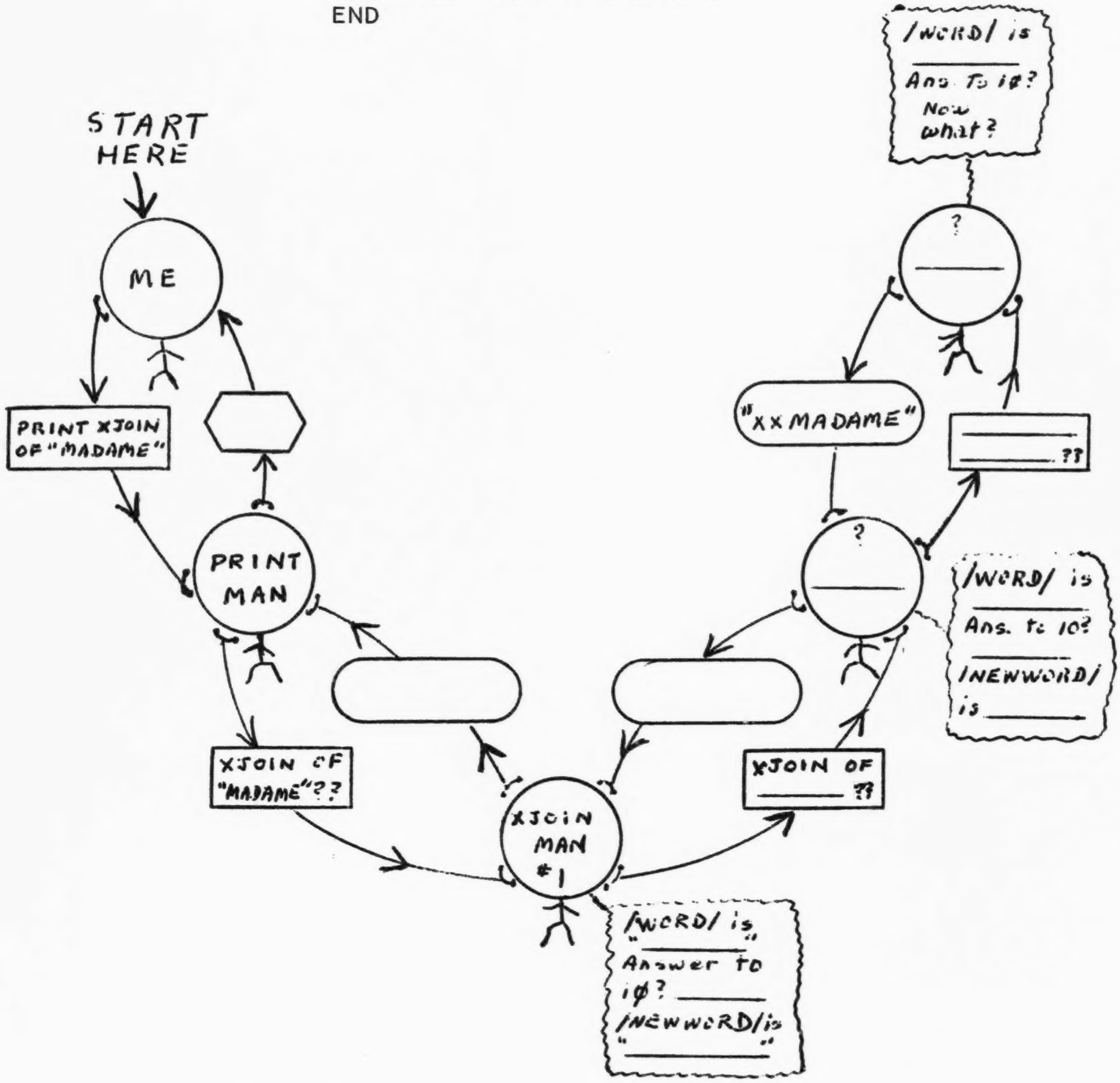
A Worked Out Example of Little Men

```
TO LASTWO /X/
10 IS COUNT OF /X/ 2
20 IF YES RETURN /X/
30 CALL
    THING:  BUTFIRST /X/
    NAME:   "X"
40 RETURN LASTWO /X/
END
```


START HERE
ME
PRINT LASTWO "TREE"
EE
PRINT MAN
Now I can do my job - I'll print EE
LASTWO "TREE"??
LASTWO OF "TREE" is "EE"
FINE! Now I can give the man who called me an answer.
LASTWO MAN #1
Hmm! Let's see what I'm supposed to do. /X/ is "TREE". The answer to 10 is NO so I won't do 20. 30 tells me that /X/ is now "REE". 40 says RETURN LASTWO "REE" but I can't do that until I find out what it is.
LASTWO OF "REE" is "EE"
LASTWO "REE"??
LASTWO MAN #2
Ah ha! /X/ is "REE". The answer to 10 is NO so I don't do 20. 30 says to call "EE" "X". 40 says RETURN LASTWO "EE" - but first I must find out what that is.
GOOD! Now I can tell the man who called me what he wanted to know.
LASTWO OF "EE" is "EE"
LASTWO "EE"??
LASTWO MAN #3
Well... /X/ is "EE" so the answer to 10 is YES. Now I have to do 20. 20 says to RETURN /X/ to the man who called me - so I'll return "EE" to him.

Complete This Little Men Picture

```
TO XJOIN /WORD/
10 IS COUNT OF /WORD/ 8
20 IF YES RETURN /WORD/
30 CALL
     THING:  WORD OF "X" AND /WORD/
     NAME:   "NEWWORD"
40 RETURN XJOIN /NEWWORD/
END
```


/WORD/ is
Ans. To 10?
Now what?
START HERE
ME
?
PRINT XJOIN OF "MADAME"
"XXMADAME"
______ ??
PRINT MAN
?
/WORD/ is
Ans. to 10?
/NEWWORD/ is ______
XJOIN OF "MADAME"??
XJOIN OF ______ ??
XJOIN MAN #1
/WORD/ is "______"
Answer to 10? ______
/NEWWORD/ is "______"

The procedure XJOIN fails for words with more than 8 letters. Noting this, a student wrote a modified procedure, XXJOIN, which is like XJOIN for inputs of 8 letters or less, but which returns the word itself as its output for any input word with more than 8 letters.

```
TO XXJOIN /L/
5 CALL
   THING:  COUNT OF /L/
   NAME:   "LENGTH"
6 CALL
   THING:  GQ OF /LENGTH/ AND "8"
   NAME:   "ANSWER"
7 IS /ANSWER/ "TRUE"
8 IF YES RETURN /L/
10 IS COUNT OF /L/ "8"
20 IF YES RETURN /L/
30 CALL
    THING:  WORD OF "X" AND /L/
    NAME:  "NW"
40 RETURN XXJOIN OF /NW/
END
```

(GQ, or GREATERQ, is a two-input predicate which outputs "TRUE" if the first input is greater than the second and "FALSE" otherwise.)

```
←P XXJOIN OF "SUPERMAN"
SUPERMAN
←P XXJOIN OF "DILLINGER"
DILLINGER
←P XXJOIN OF "1234567890"
1234567890
←P XXJOIN OF "AXE"
XXXXXAXE
←
```

Early Projects

The last weeks of the first part of the course were spent working on a few projects of somewhat larger scope. The problems were chosen for their interest to the children as well as for their value in fostering individual work and helping develop stronger resources for writing, debugging, and using programs.

One series of projects was in the area of writing interactive programs - programs in which the user communicates with the computer in the course of its operation. Examples are "conversations", question-answering programs, quizzes, etc. Another series of projects was in the area of message coding and decoding - in the childrens' parlance, "secret codes". Examples from teaching materials and student work in connection with these projects follow.

Interactive Programs

The REQUEST operation is necessary for interactive programming in LOGO. The students were introduced to REQUEST indirectly, as follows. They were told to try a procedure called LIKES before looking at the instructions comprising its definition. Typical typescripts from this use are shown next. (The student's line of typing is always preceded by an asterisk; all other lines were typed by the computer.)

```
←LIKES
TELL ME SOMETHING YOU LIKE
*BOYS
SO YOU LIKE BOYS
TELL ME SOMETHING YOU DON'T LIKE
*MR SARNO
I'M SORRY YOU DON'T LIKE MR SARNO
TELL ME SOMETHING YOU LIKE
*VACATION
SO YOU LIKE VACATION
TELL ME SOMETHING YOU DON'T LIKE
*SCHOOL
I'M SORRY YOU DON'T LIKE SCHOOL
TELL ME SOMETHING YOU LIKE
*YOU
SO YOU LIKE YOU
TELL ME SOMETHING YOU DON'T LIKE
*ME
I'M SORRY YOU DON'T LIKE ME
TELL ME SOMETHING YOU LIKE
*SCIENCE
SO YOU LIKE SCIENCE
TELL ME SOMETHING YOU DON'T LIKE
*SCIENCE
I'M SORRY YOU DON'T LIKE SCIENCE
TELL ME SOMETHING YOU LIKE
*MISSING SCHOOL WHEN I'M NOT SICK
SO YOU LIKE MISSING SCHOOL WHEN I'M NOT SICK
TELL ME SOMETHING YOU DON'T LIKE
*PEOPLE STARING AT ME
I'M SORRY YOU DON'T LIKE PEOPLE STARING AT ME
TELL ME SOMETHING YOU LIKE

          .
          .
          .

TELL ME SOMETHING YOU LIKE
*FLOWERPOWER AND LOVE
SO YOU LIKE FLOWERPOWER AND LOVE
TELL ME SOMETHING YOU DON'T LIKE
*HATE AND DEAD FLOWERS
I'M SORRY YOU DON'T LIKE HATE AND DEAD FLOWERS
TELL ME SOMETHING YOU LIKE
*NO CAVATIES
SO YOU LIKE NO CAVATIES
TELL ME SOMETHING YOU DON'T LIKE
*CANDY
I'M SORRY YOU DON'T LIKE CANDY
```

```
TELL ME SOMETHING YOU LIKE
⁑SCIENCE AND MATH (I THINK THEIR INTRESTING
SO YOU LIKE SCIENCE AND MATH (I THINK THEIR INTRESTING
TELL ME SOMETHING YOU DON'T LIKE
      .
      .
      .
TELL ME SOMETHING YOU LIKE
⁑I LIKE ICE CREAM
SO YOU LIKE I LIKE ICE CREAM
TELL ME SOMETHING YOU DON'T LIKE
⁑I DON'T LIKE SCHOOL
I'M SORRY YOU DON'T LIKE I DON'T LIKE SCHOOL
TELL ME SOMETHING YOU LIKE
⁑TELL ME SOMETHING YOU LIKE
SO YOU LIKE TELL ME SOMETHING YOU LIKE
TELL ME SOMETHING YOU DON'T LIKE
⁑I LIKE TO SKATE
I'M SORRY YOU DON'T LIKE I LIKE TO SKATE
TELL ME SOMETHING YOU LIKE
←
```

And so on. The twelve students worked at the teletypes in pairs. It took fifteen minutes before the first pair could pull themselves away from using LIKES and finally look at the procedure definition itself. (At the end of the class hour, two children were still generating their long lists of likes and dislikes. They had to be evicted from the classroom.)

Each of the children then listed the procedure LIKES to see how it worked.

```
←LIST LIKES

TO LIKES
10 PRINT "TELL ME SOMETHING YOU LIKE"
20 REQUEST "LIKE"
30 PRINT SENTENCE OF "SO YOU LIKE" AND /LIKE/
40 PRINT "TELL ME SOMETHING YOU DON'T LIKE"
50 REQUEST "NOLIKE"
60 PRINT SENTENCE OF "I'M SORRY YOU DON'T LIKE" AND /NOLIKE/
70 LIKES
END
←
```

All the children were able to work out the operation of REQUEST though none had seen this instruction previously. To demonstrate that a student did understand how LIKES worked when he said that he did, he was given the assignment of writing a procedure COPYCAT whose effect was to be as follows (the user's typing is preceded by an asterisk to distinguish it from the computer's).

```
←COPYCAT
TELL ME SOMETHING
*I GO LOGO
I GO LOGO
TELL ME SOMETHING
*I LOVE YOU
I LOVE YOU
TELL ME SOMETHING
      .
      .
      .
```

A typical COPYCAT program:

```
TO COPYCAT
10 PRINT "TELL ME SOMETHING"
20 REQUEST "SOMETHING"
30 PRINT /SOMETHING/
40 COPYCAT
END
```

After showing that they could write COPYCAT, most of the children continued on their own to write other interactive procedures patterned after these models. Examples of some of these are shown next (in each case the procedure is listed and then run by a student).

```
TO FOOD
10 PRINT "TELL ME THE FOOD THAT YOU LIKE BEST"
20 REQUEST "FOOD"
30 PRINT SENTENCE OF "SO YOU LIKE" AND SENTENCE OF /FOOD/
    AND "BEST"
40 PRINT "TELL ME SOMETHING YOU DO NOT LIKE"
50 REQUEST "NOLIKE"
60 PRINT SENTENCE OF "I DO NOT LIKE THAT EITHER" (A bug here)
END
```

```
←FOOD
TELL ME THE FOOD THAT YOU LIKE BEST
∷STEAK
SO YOU LIKE STEAK BEST
TELL ME SOMETHING YOU DO NOT LIKE
∷HAMBURG
THERE IS SOMETHING MISSING ON THIS LINE.
I WAS AT LINE 60 IN FOOD.
←
```

(The program stopped because SENTENCE needs <u>two</u> inputs)

```
TO ME
10 PRINT "YOUR NAME"
20 REQUEST "ME"
30 PRINT SENTENCE OF /ME/ AND "IS SILLY"
40 ME
END
```

```
←ME
YOUR NAME
∷HENRY
HENRY IS SILLY
YOUR NAME
∷HENRIETTA
HENRIETTA IS SILLY
YOUR NAME
∷BILLY
BILLY IS SILLY
      .
      .
      .
```

```
TO AGE
10 PRINT "TELL ME YOUR AGE"
20 REQUEST "AGE"
30 PRINT SENTENCE OF "SO YOU AGE" AND /AGE/
40 PRINT "TELL ME MY AGE"
50 REQUEST "AGE"
60 PRINT SENTENCE OF "SO I AGE" AND /AGE/
70 AGE
END
```

(The English sentence structure is something less than perfect; but the formal structure of the program is correct, and, for this student, this program was a significant intellectual achievement.)

```
←AGE
TELL ME YOUR AGE
*56
SO YOU AGE 56
TELL ME MY AGE
*78
SO I AGE 78
TELL ME YOUR AGE
*2
SO YOU AGE 2
TELL ME MY AGE
*1234
SO I AGE 1234
TELL ME YOUR AGE
*765432876543545676
SO YOU AGE 765432876543545676
TELL ME MY AGE
      .
      .
      .
```

```
←TO SING
>10 PRINT "WHAT SONG DO YOU WANT TO HEAR?"
>20 REQUEST "E"
>30 PRINT "I DON'T KNOW THAT ONE PLEASE HUM A FEW BARS FIRST"
>40 PRINT "I CAN'T HEAR YOU HUM LOUDER"
>50 PRINT "O K I'LL TRY IT NOW"
>60 PRINT S S S S S S /BELL/ /BELL/ /BELL/ /BELL/ /BELL/
     /BELL/ /BELL/
>70 PRINT "THAT'S THE BEST I CAN DO"
>END
SING DEFINED
←
```

(The effect of the instruction in Line 60, which prints an invisible but audible seven-word sentence, is to ring the teletype bell seven times.)

```
←SING
WHAT SONG DO YOU WANT TO HEAR?
*AMERICA
I DON'T KNOW THAT ONE PLEASE HUM A FEW BARS FIRST
I CAN'T HEAR YOU HUM LOUDER
O K I'LL TRY IT NOW
                        (At this point it rings the bells)
THAT'S THE BEST I CAN DO
←
```

```
←SING
WHAT SONG DO YOU WANT TO HEAR?
※ANY SONG
I DON'T KNOW THAT ONE PLEASE HUM A FEW BARS FIRST
I CAN'T HEAR YOU HUM LOUDER
O K I'LL TRY IT NOW
                                  (Seven bells again)
THAT'S THE BEST I CAN DO
←
```

In some instances, these procedures were the first ones wholly conceived by the children. All the children were very serious about this work, even when writing procedures that might appear silly or funny. For some of the children getting a procedure of this kind to work required (what was for them) a formidable intellectual effort.

In the next phase of interactive programming, work was done in connection with various quiz programs. Four partially developed quiz programs - PREACH, LOGOPART, SPORTQUIZ, and COMICSTRIPQUIZ - were tried by the students. Sample runs with these programs follow next. (The student's typing is underscored to distinguish it from the computer's.)

```
(A FUNNY PROGRAM)
←PREACH
THINK OF A NUMBER.  TYPE IT WHEN YOU SEE ※.
※12345
YOUR NUMBER IS ODD.
DO YOU WANT TO KNOW MORE?  TYPE YES OR NO.
※YES
IT'S BIGGER THAN A THOUSAND.
DO YOU WANT TO KNOW MORE?  TYPE YES OR NO.
※YES
YOUR NUMBER IS LESS THAN 12350
DO YOU WANT TO KNOW MORE?  TYPE YES OR NO.
※YES
TWICE YOUR NUMBER IS EVEN.
DO YOU WANT TO KNOW MORE?  TYPE YES OR NO.
※YES
```

(continued)

```
SORRY, THAT'S ALL I KNOW.  I'M A PRETTY DUMB COMPUTER.  IF
YOU WANT TO KNOW MORE YOU'D BETTER START ME AGAIN OR GIVE
ME SOME NEW THINGS TO DO.

I GUESS I WON'T WAIT FOR YOU TO START ME.
THINK OF A NUMBER.  TYPE IT WHEN YOU SEE ⁙.
⁙34
YOUR NUMBER IS EVEN
DO YOU WANT TO KNOW MORE?  TYPE YES OR NO.
⁙YES
THAT'S A SMALL NUMBER.
DO YOU WANT TO KNOW MORE?  TYPE YES OR NO.
⁙NO



←LOGOPART

TYPE ANY WORD WHEN YOU SEE ⁙
⁙KP
I'LL CALL KP "ANYWORD"

TYPE FIRST OF /ANYWORD/
⁙K
GOOD FOR YOU!

TYPE LAST OF /ANYWORD/
⁙O
SORRY, LAST OF /ANYWORD/ IS P

TYPE IN /ANYWORD/
⁙OK
TYPE IN BF OF /ANYWORD/
⁙K
YOU ARE BRILLIANT

I'LL BET YOU CAN ADD MANY MORE INSTRUCTIONS TO THIS PROGRAM.
←
```

```
←SPORTQUIZ

ARE YOU A SPORTS FAN?  DO YOU LIKE BASEBALL?  FOOTBALL?
HOCKEY?  BASKETBALL?  HERE ARE SOME QUESTIONS TO GIVE YOU
A CHANCE TO SHOW OFF WHAT YOU KNOW.

WHAT TEAM WON THE AMERICAN LEAGUE PENNANT THIS YEAR? (GIVE
THE NAME OF THE CITY FIRST AND THEN THE TEAM NAME. EXAMPLE:
CHICAGO CUBS)
*CARNALS
STRIKE ONE.  THE DETROIT TIGERS WON THE 1968 AMERICAN
LEAGUE PENNANT.

DID THE ST. LOUIS CARDINALS WIN THE WORLD SERIES THIS YEAR?
ANSWER YES OR NO.
*NO
BASE HIT

DO THE BOSTON PATRIOTS PLAY FOOTBALL IN THE NFL OR THE AFL?
*NO
FIVE YARD PENALTY.  THE BOSTON PATRIOTS PLAY IN THE
AMERICAN FOOTBALL LEAGUE -- THE AFL.

I THINK YOU PROBABLY KNOW A LOT MORE ABOUT SPORTS THAN I DO.
WHY DON'T YOU ADD SOME QUESTIONS (AND ANSWERS, OF COURSE)
TO THIS QUIZ?  THEN YOU CAN TRY YOUR VERSION.

←COMICSTRIPQUIZ

ARE YOU A FAITHFUL READER OF THE COMICS?  DO YOU CHUCKLE
WHEN YOUR FAVORITE CHARACTER GETS INTO A FUNNY SITUATION?
DO YOU LIKE THE ADVENTURE COMICS?  THE DETECTIVE COMICS?
THE TEENAGER COMICS?  HERE ARE SOME QUESTIONS TO SEE HOW
LOYAL A COMIC STRIP READER YOU ARE?

THE RED BARON WAS A GERMAN WORLD WAR I HERO.  WHO FIGHTS
THE RED BARON IN THE COMIC STRIPS?
*H
IT'S SNOOPY WHO SAYS 'CURSE YOU, RED BARON'

ROBIN IS THE BOY ASSOCIATE OF WHAT COMIC STRIP CHARACTER?
*BATMAN
HOLY COMPUTER!  YOU'RE RIGHT.

WHO WAS THE FAMOUS CARTOONIST AND PRODUCER WHO CREATED SUCH
CHARACTERS AS MICKEY MOUSE, DONALD DUCK, AND PLUTO?
*W
THE GREAT WALT DISNEY IS THE NAME YOU WANT.
```

```
NOW IT'S YOUR TURN TO WRITE SOME QUESTIONS ABOUT YOUR
FAVORITE COMIC STRIP CHARACTERS.  ADD THEM TO THIS QUIZ
SO WE CAN TRY THE QUIZ OUT ON EACH OTHER.
←
```

The students then listed each of the quiz programs to see how it was constructed. Here, for example, is a listing of the procedure COMICSTRIPQUIZ along with the three procedures that it uses - Q1, Q2, and Q3.

```
TO COMICSTRIPQUIZ
10 PRINT ""
20 PRINT ""
30 PRINT ""
40 PRINT "ARE YOU A FAITHFUL READER OF THE COMICS?  DO YOU
   CHUCKLE WHEN YOUR FAVORITE CHARACTER GETS INTO A FUNNY
   SITUATION?  DO YOU LIKE THE ADVENTURE COMICS?  THE
   DETECTIVE COMICS?  THE TEENAGER COMICS?  HERE ARE SOME
   QUESTIONS TO SEE HOW LOYAL A COMIC STRIP READER YOU ARE?"
42 PRINT ""
45 PRINT ""
50 PRINT Q1
55 PRINT ""
58 PRINT ""
60 PRINT Q2
62 PRINT ""
65 PRINT ""
70 PRINT Q3
75 PRINT ""
80 PRINT ""
85 PRINT "NOW IT'S YOUR TURN TO WRITE SOME QUESTIONS ABOUT
   YOUR FAVORITE COMIC STRIP CHARACTERS.  ADD THEM TO THIS
   QUIZ SO WE CAN TRY THE QUIZ OUT ON EACH OTHER."
END

TO Q1
10 PRINT "THE RED BARON WAS A GERMAN WORLD WAR I HERO.  WHO
   FIGHTS THE RED BARON IN THE COMIC STRIPS?"
15 REQUEST "X"
20 IS /X/ "SNOOPY"
25 IF YES RETURN "RIGHT YOU ARE."
30 IF NO RETURN "IT'S SNOOPY WHO SAYS 'CURSE YOU, RED BARON'"
END
```

```
TO Q2
10 PRINT "ROBIN IS THE BOY ASSOCIATE OF WHAT COMIC STRIP
   CHARACTER?"
20 REQUEST "Y"
30 IS /Y/ "BATMAN"
40 IF YES RETURN "HOLY COMPUTER!  YOU'RE RIGHT."
50 IF NO PRINT "HOLY SMOKES!  IT'S BATMAN"
END

TO Q3
10 PRINT "WHO WAS THE FAMOUS CARTOONIST AND PRODUCER WHO
   CREATED SUCH CHARACTERS AS MICKEY MOUSE, DONALD DUCK,
   AND PLUTO?"
20 REQUEST "WHO"
30 IS /WHO/ "WALT DISNEY"
40 IF YES RETURN "YOU'RE AN AUTHORITY ON CARTOONISTS, I SEE."
50 IF NO RETURN "THE GREAT WALT DISNEY IS THE NAME YOU WANT."
END
```

The students then made up their own questions and answers in the form of procedures. Here is a procedure made for use with COMICSTRIPQUIZ.

```
TO Q4
10 PRINT "WHO IS HAWK'S PARTNER?"
20 REQUEST "WHO"
30 IS /WHO/ "DOVE"
40 IF YES RETURN "GEE WILLIKERS, THATS RIGHT"
50 IF NO RETURN "DOVE IS THE NAME, JOKES ARE MY GAME"
END
```

COMICSTRIPQUIZ was then edited to incorporate this new question procedure, Q4.

```
←EDIT COMICSTRIPQUIZ
>90 PRINT ""
>95 PRINT ""
>100 PRINT Q4
>END
COMICSTRIPQUIZ DEFINED
←
```

When COMICSTRIPQUIZ was run, the following exchange occurred after those shown in the previous run.

```
WHO IS HAWK'S PARTNER?
*ROBIN
DOVE IS THE NAME, JOKES ARE MY GAME
```

All the quiz programs shown use preprogrammed questions and answers. In later phases, students wrote procedures for generating the questions and answers as these were needed in the course of the quiz. This work was done in conjunction with a major project on equations, as part of the algebra course material. It is discussed later (in the section titled Algebra Teaching Sequence).

Message Coding and Decoding Programs

The work on message coding programs started with a procedure called SCRAMBLE, which rotates the letters of a word one position to the left (circularly - the leftmost letter becomes the rightmost one). SCRAMBLE and its associated decoding procedure, UNSCRAMBLE, were introduced with bugs. A copy of the students' assignment to debug them follows.

The students discovered that the roles of SCRAMBLE and UNSCRAMBLE could be reversed, i.e., that UNSCRAMBLE could be used for coding a word and that SCRAMBLE would then function properly as the associated decoder.

BUGS, BUGS, BUGS!!

Here are two procedures that don't quite work. Find the bugs. The first procedure is called SCRAMBLE.

```
TO SCRAMBLE /WORD/
10 CALL
     THING:  FIRST OF /WORD/
     NAME:   "FRONT"
20 CALL
     THING:  BUTFIRST OF /WORD/
     NAME:   "BACK"
30 RETURN WORD OF /FRONT/ AND /BACK/
END
```

This procedure was supposed to scramble the word by putting the first letter at the end of the word. Here is what happened.

```
←PRINT SCRAMBLE OF "DOOR"    I wanted this procedure to return
DOOR                              OORD
←PRINT SCRAMBLE OF "PRESS"
PRESS                             RESSP
←
```

Once the procedure SCRAMBLE worked, UNSCRAMBLE was supposed to put the word back together again.

```
TO UNSCRAMBLE /MESS/
10 CALL
     THING:  LAST OF /MESS/
     NAME:   "FRONT"
20 CALL
     THING:  BUTLAST OF /MESS/
     NAME:   "BACK"
30 CALL
     THING:  WORD OF /FRONT/ AND /BACK/
     NAME:   "NEWWORD"
40 RETURN /MESS/
END UNSCRAMBLE
```

Here is what happened. I wanted this procedure to return

```
←PRINT UNSCRAMBLE OF "OORD"
OORD                              DOOR
←PRINT UNSCRAMBLE OF "RESSP"
RESSP                             PRESS
←
```

Can you fix these two procedures?

The main project of the series on "secret codes", the Gibberish project, was suggested by the students. They were all fluent speakers of Gibberish (pronounced Jibberish). They provided the rules for translating English words to simple Gibberish: if the first letter is a consonant, insert the letters ITHAG after the first letter; if the first letter is a vowel, prefix the entire word with the letters ITHAG. (These are very similar to Pig Latin rules.) Thus, DOG becomes DITHAGOG, and CAT becomes CITHAGAT, but AT becomes ITHAGAT and I becomes ITHAGI. The goal was to write procedures for translating English sentences to Gibberish sentences and vice-versa.

The first task was to write a procedure, called GIB, for performing the first of the two translation rules. GIB has a single input, the word /OLDWORD/. Its output is a word constructed from three parts - the first letter of the input, the literal "ITHAG", and the BUTFIRST of the input.

```
←TO GIB /OLDWORD/
>10 CALL
     THING:  FIRST OF /OLDWORD/
     NAME:  "ONE"
>20 CALL
     THING:  "ITHAG"
     NAME:  "TWO"
>30 CALL
     THING:  BUTFIRST OF /OLDWORD/
     NAME:  "THREE"
>40 RETURN WORD OF /ONE/ AND WORD OF /TWO/ AND /THREE/
>END
GIB DEFINED

←PRINT GIB OF "CAT"
CITHAGAT
←PRINT GIB OF "AT"
AITHAGT
←PRINT GIB OF "A"
AITHAG
←
```

Note that GIB has the same effect on words beginning with vowels as it has on words beginning with consonants.

The second task was to write a procedure, called IB, for performing the second of the two translation rules.

```
←TO IB /OLDWORD/
>10 CALL
     THING:  "ITHAG"
     NAME:  "ONE"
>20 CALL
     THING:  /OLDWORD/
     NAME:  "TWO"
>30 RETURN WORD OF /ONE/ AND /TWO/
>END
IB DEFINED

←PRINT IB OF "A"
ITHAGA
←PRINT IB OF "AT"
ITHAGAT
←PRINT IB OF "CAT"
ITHAGCAT
←
```

Like GIB, IB works indifferently on all words.

The third task was to write a test procedure to decide which of the two procedures (IB or GIB) is to be performed on a given input: the test is whether or not the first letter of the input is a vowel. The students wrote test procedures like the following.

```
←TO VOWEL /LETTER/
>10 IS /LETTER/ "A"
>20 IF YES RETURN "YES"
>30 IS /LETTER/ "E"
>40 IF YES RETURN "YES"
>50 IS /LETTER/ "I"
>60 IF YES RETURN "YES"
>70 IS /LETTER/ "O"
>80 IF YES RETURN "YES"
>90 IS /LETTER/ "U"
>100 IF YES RETURN "YES"
>110 RETURN "NO"
>END
VOWEL DEFINED
←
```

Next, students wrote a procedure, called SUPERGIB, for translating any English word. They usually started with a faulty procedure like this.

```
TO SUPERGIB /ANYWORD/
10 IS FIRST OF /ANYWORD/ VOWEL
20 IF YES RETURN IB OF /ANYWORD/
30 IF NO RETURN GIB OF /ANYWORD/
END
```

Line 10 is better English than LOGO. VOWEL needs to have an input. Also, its output must be one of the two words "YES" or "NO", not a letter. Line 10 is correctly rewritten as follows:

```
10 IS VOWEL OF FIRST OF /ANYWORD/ "YES"
```

Now SUPERGIB works.

```
←PRINT SUPERGIB OF "JIM"
JITHAGIM
←PRINT SUPERGIB OF "AMY"
ITHAGAMY
←
```

The students could translate two-word sentences in the following way.

```
←PRINT SENTENCE OF SUPERGIB OF "JIM" AND SUPERGIB OF "AMY"
JITHAGIM ITHAGAMY
←
```

They were shown a general procedure for translating sentences of arbitrary length into Gibberish.

```
←TO GIBBERISH /SENT/
>10 IS /SENT/ /EMPTY/
>20 IF YES RETURN /EMPTY/
>30 CALL
     THING:  SUPERGIB OF FIRST OF /SENT/
     NAME:  "GIBWORD"
>40 RETURN SENTENCE OF /GIBWORD/ AND GIBBERISH OF BUTFIRST
    OF /SENT/
>END
GIBBERISH DEFINED
←
```

```
←PRINT GIBBERISH OF "THIS DOES IT"
TITHAGHIS DITHAGOES ITHAGIT
←
```

This procedure would have been too difficult for the students to write at this stage of their development. Some of them, however, after seeing the procedure GIBBERISH could have written an UNGIBBERISH for turning a Gibberish sentence into an English sentence.

The students' last assignment for this project was a slightly easier one - to write a procedure UNGIB for undoing SUPERGIB, i.e., for turning a Gibberish word into an English word. Here is the UNGIB procedure of one of the students.

```
TO UNGIB /M/
8 IS FIRST OF /M/ "I"
9 IF YES RETURN BUTFIRST BUTFIRST BUTFIRST BUTFIRST BUTFIRST
  OF /M/
10 CALL
     THING:  FIRST OF /M/
     NAME:  "FRONT"
20 CALL
     THING:  BUTFIRST BUTFIRST BUTFIRST BUTFIRST BUTFIRST
             BUTFIRST OF /M/
     NAME:  "BACK"
30 CALL
     THING:  WORD OF /FRONT/ AND /BACK/
     NAME:  "NEWWORD"
40 RETURN /NEWWORD/
END

←P UNGIB "GITHAGOOD"
GOOD
←P UNGIB "GITHAGIRL"
GIRL
←P UNGIB "YITHAGOU"
YOU
←P UNGIB "DITHAGID"
DID
←P UNGIB "ITHAGIT"
IT
←
```

In this procedure line 8 tests whether the input begins with "I". If it does (this corresponds to English words beginning with a vowel), the input is necessarily prefixed by "ITHAG" and the output is obtained (on line 9) by striking off those five letters from the input. In the other case, the output is the word made by joining the first letter of the input with what remains of the input after the first six letters are removed (letters 2 through 6 must be I,T,H,A,G). Note that the sequence of test runs at the end spell out the words GOOD GIRL YOU DID IT (a hidden message). She obviously felt that she had done a hard job well.

The project following Gibberish was Pig Latin. The procedures for translating English into Pig Latin are very similar to those for Gibberish. With Pig Latin, however, the students were introduced to a different VOWEL procedure which uses the following more general procedure for finding whether or not a letter is contained in a word. (It outputs "YES" or "NO" accordingly.)

```
TO CONTAINS /LETTER/ AND /WORD/
10 IS /WORD/ /EMPTY/
20 IF YES RETURN "NO"
30 IS /LETTER/ FIRST OF /WORD/
40 IF YES RETURN "YES"
50 IF NO RETURN CONTAINS OF /LETTER/ AND BUTFIRST OF /WORD/
END
```

The procedure works as follows. It tests (in line 30) to see if the letter it is checking for is the same as the first letter of the word in question. If it is, the procedure outputs "YES" (line 40). If it is not, the procedure is repeated, this time (line 50) testing the letter against the butfirst of the word (i.e., the word obtained by removing the first letter of the current word). If the word becomes empty (i.e., no more letters), the procedure outputs "NO". VOWEL can now be written as a special case of CONTAINS in which the word being searched is the word of all vowel letters, "AEIOU".

```
TO VOWEL /LETTER/
10 RETURN CONTAINS OF /LETTER/ AND "AEIOU"
END
```

The value of having this more general procedure, CONTAINS, was shown by the ease with which some other test procedures could be written with it. Thus, the procedure EVEN, which tests whether or not a number is even.

```
TO EVEN /NUMBER/
10 RETURN CONTAINS OF LAST OF /NUMBER/ AND "02468"
END
```

Extensions of the use of CONTAINS for selecting specified (sometimes randomly specified) letters from words or words from sentences were made in subsequent projects on generating sentences and algebraic equations.

4.3 Algebra Materials

In the second part of the course, which covered a period of about six months, the objective was to use LOGO in the teaching of mathematics, including specific content in arithmetic and algebra. The effort was focused on exploratory development of an elaborated, coherent curriculum.

Samples of the teaching materials used in four sequences, including associated student work, are presented next.

Sequences and Oscillators

The second part of the course started with a unit on the generation of number sequences. The material was chosen and organized to show the basic structure of simple iterative processes. An iteration is built up out of four separable functional parts - a CHECK to determine whether or not the iteration is complete, an ACTION to be carried out each time the iteration is performed (i.e., at each round), the PREPARATION of inputs for the next round of the iteration, and the RECURSION, which calls for the execution of the next round.

The unit comprised four assignments. The first one studies a procedure called GODOWN to generate integers in descending sequence from any starting number to 1. The second one treats a related procedure called GOUP to generate integers in ascending sequence between two prescribed numbers. The student is given the skeleton of the GOUP procedure and he is supposed to complete it. The third assignment concerns a modified GODOWN procedure; this one completely analogous to GOUP. The student is given a modified GODOWN which has bugs. He is to debug it. The last assignment is for the student to write a procedure UPANDDOWN which is the grand summing-up of the preceding ones.

Sequences - Assignment 1

```
TO GODOWN /NUMBER/
10 IS /NUMBER/ GREATER OF /NUMBER/ AND 1          (CHECK)
20 IF NO RETURN
30 PRINT /NUMBER/                                 (ACTION)
40 CALL                                           (PREPARATION)
     THING:  DIFFERENCE OF /NUMBER/ AND 1
     NAME:  "NEWNUMBER"
50 RETURN GODOWN OF /NEWNUMBER/                   (STARTS NEW
                                                   ROUND)
```

Line 10 checks whether GODOWN is done.

It checks whether /NUMBER/ is greater than or equal to 1.

GODOWN is done when /NUMBER/ is less than 1.

If line 30 became line 40 and line 40 became line 30:

```
30 CALL                                           (PREPARATION)
     THING:  DIFFERENCE OF /NUMBER/ AND 1
     NAME:  "NEWNUMBER"
40 PRINT /NUMBER/                                 (ACTION)
```

Would GODOWN act differently? NO

To answer the next question you have to play computer with GODOWN.

What happens if the action line (now Line 40) is

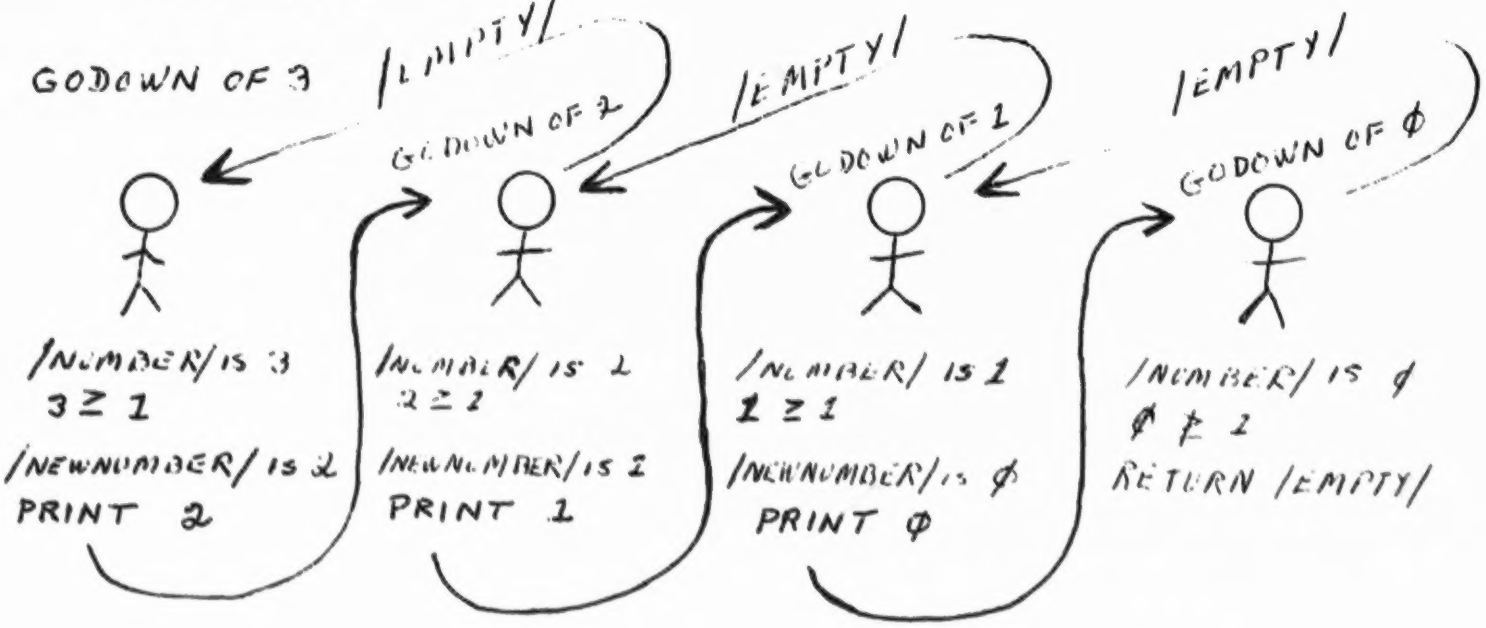
```
     40 PRINT /NEWNUMBER/                                  (ACTION)
```

GODOWN now looks like

```
TO GODOWN /NUMBER/
10 IS /NUMBER/ GREATER OF /NUMBER/ AND 1                   (CHECK)
20 IF NO RETURN
30 CALL                                                    (PREPARATION)
     THING:  DIFFERENCE OF /NUMBER/ AND 1
     NAME:  "NEWNUMBER"
40 PRINT /NEWNUMBER/                                       (ACTION)
50 RETURN GODOWN OF /NEWNUMBER/                            (STARTS NEW
                                                            ROUND)
```

PLAY COMPUTER

GODOWN OF 3


GODOWN OF 3
/EMPTY/
GODOWN OF 2
/EMPTY/
GODOWN OF 1
/EMPTY/
GODOWN OF 0
/NUMBER/ IS 3
3 ≥ 1
/NEWNUMBER/ IS 2
PRINT 2
/NUMBER/ IS 2
2 ≥ 1
/NEWNUMBER/ IS 1
PRINT 1
/NUMBER/ IS 1
1 ≥ 1
/NEWNUMBER/ IS 0
PRINT 0
/NUMBER/ IS 0
0 ≱ 1
RETURN /EMPTY/

Try GODOWN with the following inputs

3

5

7

Sequences - Assignment 2

GOUP is to print its first input until it is greater than its second input. Before each round a new first input must be prepared by adding 1 to the present first input.

Fill in the blanks.

```
TO GOUP /ACTIONNUMBER/ AND /TOPNUMBER/
10 IS /TOPNUMBER/ GREATER OF /ACTIONNUMBER/ AND /TOPNUMBER/(CHECK)
20 IF NO RETURN
30 CALL                                               (PREPARATION)
     THING:  SUM  OF /ACTIONNUMBER/ AND 1
     NAME:  "NEWNUMBER"
40 PRINT /ACTIONNUMBER/                               (ACTION)
50 RETURN GOUP OF /NEWNUMBER/ AND /TOPNUMBER/         (STARTS NEW
                                                       ROUND)
```

Line 10 is the check line.

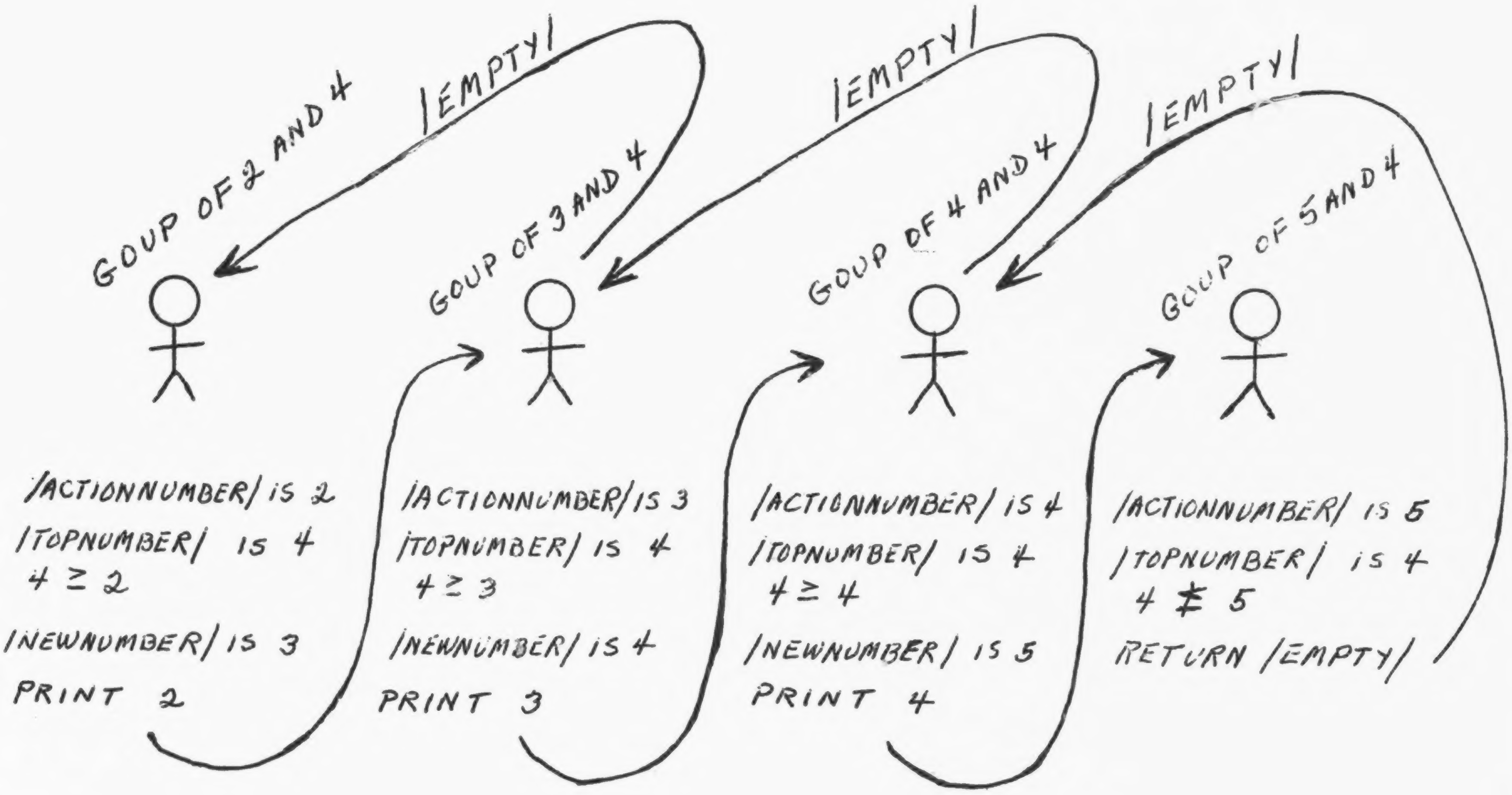
It checks whether /TOPNUMBER/ is greater than or equal to /ACTIONNUMBER/.

When /TOPNUMBER/ is less than /ACTIONNUMBER/ GOUP returns /EMPTY/.

Line 30 is the ACTION line. It ADDS 1 to /ACTIONNUMBER/.

To check your answers to the questions above, PLAY COMPUTER with GOUP.

```
←GOUP OF 2 AND 4
2
3
4
←
```


GOUP OF 2 AND 4
/EMPTY/
GOUP OF 3 AND 4
/EMPTY/
GOUP OF 4 AND 4
/EMPTY/
GOUP OF 5 AND 4
/ACTIONNUMBER/ IS 2
/TOPNUMBER/ IS 4
4 ≥ 2
/NEWNUMBER/ IS 3
PRINT 2
/ACTIONNUMBER/ IS 3
/TOPNUMBER/ IS 4
4 ≥ 3
/NEWNUMBER/ IS 4
PRINT 3
/ACTIONNUMBER/ IS 4
/TOPNUMBER/ IS 4
4 ≥ 4
/NEWNUMBER/ IS 5
PRINT 4
/ACTIONNUMBER/ IS 5
/TOPNUMBER/ IS 4
4 ≱ 5
RETURN /EMPTY/

Now - at the terminal - get the incomplete procedure GOUP. Edit it by retyping the lines which you filled in above. Then try it.

Sequences - Assignment 3

GOUP is a procedure which requires two inputs - a starting number and a stopping number.
Change GODOWN so that it requires two inputs - a starting number and a stopping number.

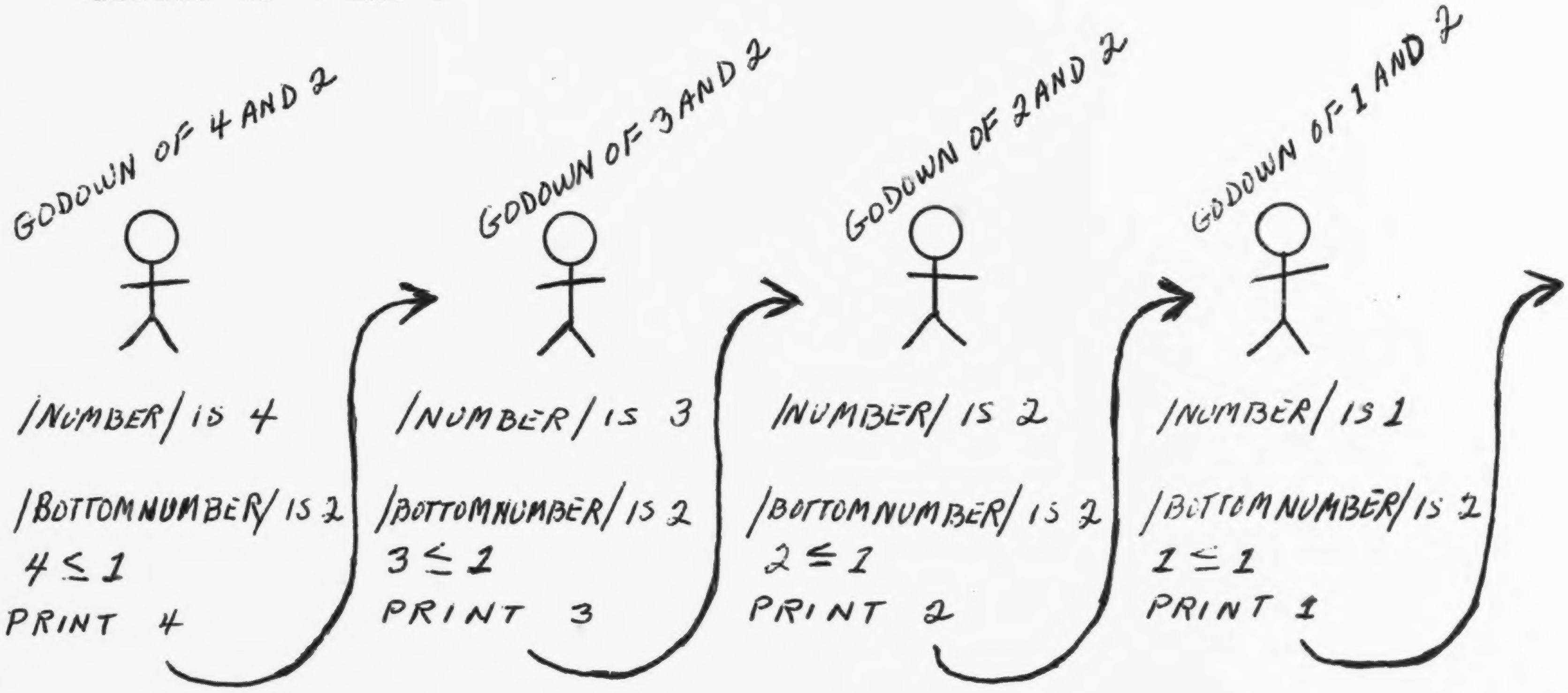
```
←GODOWN 5 AND 2
5
4
3
2
←
```

```
←LIST GODOWN
TO GODOWN /NUMBER/
10 IS /NUMBER/ GREATER OF /NUMBER/ AND 1          (CHECK)
20 IF NO RETURN
30 CALL                                            (PREPARATION)
     THING:  DIFFERENCE OF /NUMBER/ AND 1
     NAME:  "NEWNUMBER"
40 PRINT /NUMBER/                                  (ACTION)
50 RETURN GODOWN OF /NEWNUMBER/                    (STARTS NEW
                                                    ROUND)
```

How many inputs will the new GODOWN require? 2
Which lines in GODOWN need to be changed? 10, 50, TITLE
The CHECK line and line 50 which starts the next round need to be changed.
Fill in:
TITLE TO GODOWN /NUMBER/ AND /BOTTOMNUMBER/
50 RETURN GODOWN OF /NEWNUMBER/ AND /BOTTOMNUMBER/

Now that these changes have been made will GODOWN do the right thing? NO
To answer this question play computer.

```
←GODOWN OF 4 AND 2
```


GODOWN OF 4 AND 2
/NUMBER/ IS 4
/BOTTOMNUMBER/ IS 2
4 ≤ 1
PRINT 4
GODOWN OF 3 AND 2
/NUMBER/ IS 3
/BOTTOMNUMBER/ IS 2
3 ≤ 1
PRINT 3
GODOWN OF 2 AND 2
/NUMBER/ IS 2
/BOTTOMNUMBER/ IS 2
2 ≤ 1
PRINT 2
GODOWN OF 1 AND 2
/NUMBER/ IS 1
/BOTTOMNUMBER/ IS 2
1 ≤ 1
PRINT 1

There is a bug. What is wrong?

Is GODOWN performing the right action? YES

Is the check line correct? NO

When should GODOWN be done? WHEN /NUMBER/ IS LESS THAN /BOTTOMNUMBER/

GODOWN is done when /NUMBER/ is <u>no</u> longer greater than or equal to /BOTTOMNUMBER/

GODOWN must be edited again.

Line 10 must be changed.

The second input to GODOWN must be used in the check line instead of 1 .

Now EDIT GODOWN at the terminal.

Try it with the following pairs of inputs.

5 AND 2

3 AND 1

101 AND 97

Sequences - Assignment 4

Write a procedure UPANDDOWN which will swing down and up between its two inputs.

NAME OF PROCEDURE: TO UPANDDOWN /BOTTOM/ /TOP/

EXAMPLE OF USE:

UPANDDOWN 2 AND 5

```
5
4
3
2
3
4
5
4
3
2
3
4
5
4
.
.
.
```

UPANDDOWN will continue until we depress the BREAK key. How can we modify the procedure so UPANDDOWN stops after a specified number of complete swings?

The concepts introduced in the unit on sequences were consolidated and extended in the following unit. The material is part of a larger sequence on oscillators planned for subsequent teaching. It begins with procedures for generating simple oscillatory patterns. The procedure SAW is a natural continuation of number sequencing procedures like UPANDDOWN.

Our next unit is about procedures which draw designs. The first pattern-making procedure we shall study is called SAW.

NAME OF PROCEDURE: TO SAW /COUNTER/ AND /LIMIT/

INPUTS: /COUNTER/ - any numeral

/LIMIT/ - maximum word size

EXAMPLE:

```
←SAW OF 1 AND 3
X
 X
XXX
X
 X
XXX
X
 X
XXX
X
 X
XXX
X
 .
 .
 .

←SAW OF 4 AND 6
   X
    X
XXXXXX
X
 X
  X
   X
    X
XXXXXX
X
 X
  X
   X
    X
XXXXXX
X
 .
 .
 .
```

The pattern is like a saw's teeth. Two kinds of words make up the pattern. One kind is a "solid" word like:

```
XXXXXXXX
```

the other is a single letter preceded by blanks (a margin)

```
       X
```

SAW needs separate procedures to make each kind of word.

```
EXAMPLES:                       NAME OF PROCEDURE:

PRINT REPEAT OF "X" AND 6       TO REPEAT /WORD/ AND /LENGTH/
XXXXXX

PRINT REPEAT OF "A" AND 10
AAAAAAAAAA

PRINT PLOT OF "X" AND 6         TO PLOT /WORD/ AND /LENGTH/
     X

PRINT PLOT OF "A" AND 10
         A
```

SAW relies on its own CHOOSE procedure to decide when to use PLOT and when to use REPEAT.

NAME OF PROCEDURE: TO CHOOSE /NUMBER/ AND /LIMIT/

We find out how CHOOSE makes its decision by looking at SAW's output. This time we number the lines.

```
SAW OF 1 AND 4
1  X
2   X
3    X
4 XXXX
5 X
6  X
7   X
8 XXXX
9 X
   .
   .
   .
```

Lines 4 and 8 use REPEAT. The other lines use PLOT. If we think of /COUNTER/ as the line number, we can see that when /COUNTER/ is 4 or 8 it is divisible by /LIMIT/ which is 4. Whenever /COUNTER/

divided by /LIMIT/ has a remainder of 0, CHOOSE uses REPEAT. Otherwise, CHOOSE gives PLOT the remainder as its word length.

The other procedure needed to make up this design is

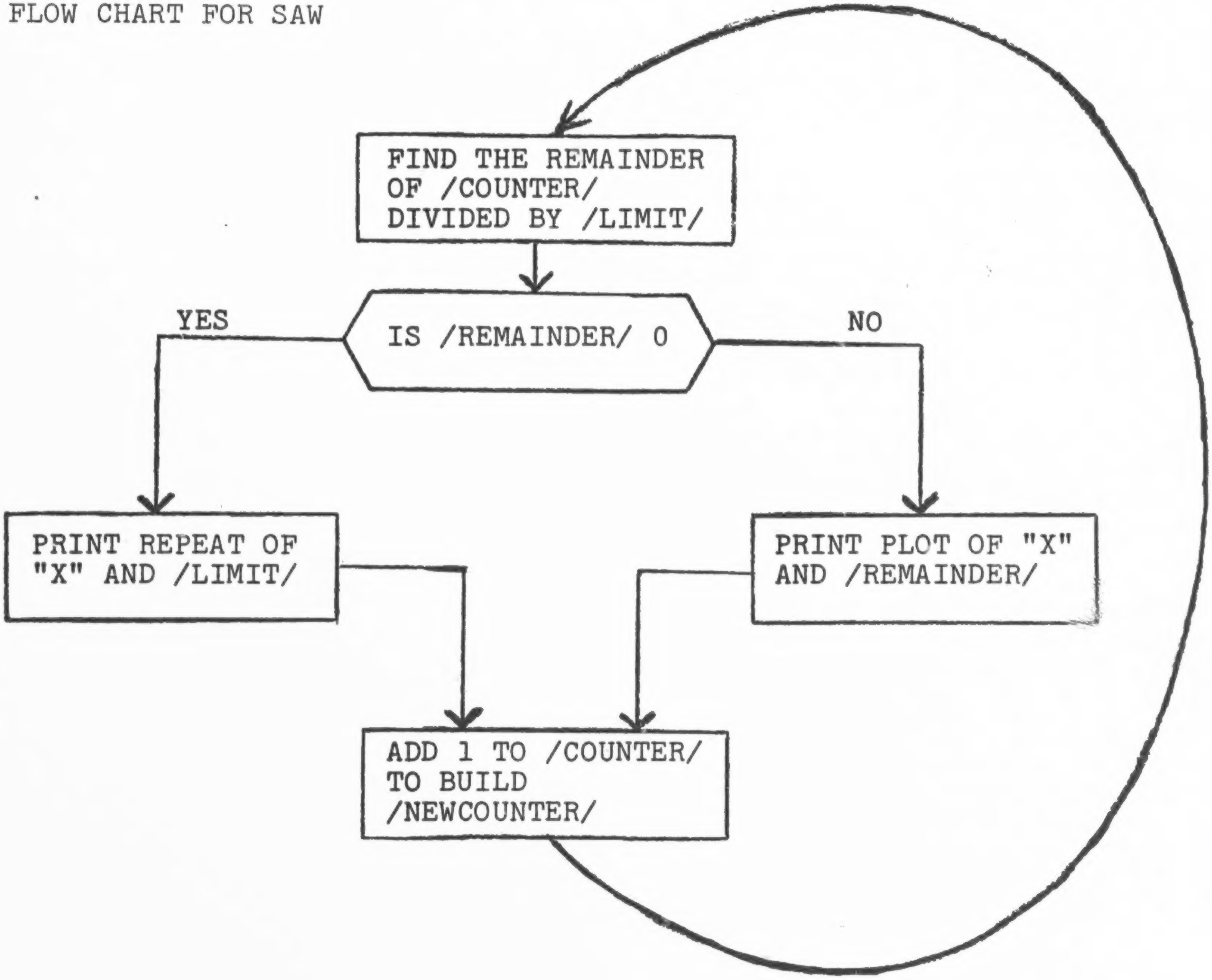
NAME OF PROCEDURE: TO REM /NUMBER/ AND /DIVISOR/

This procedure finds the remainder of /NUMBER/ divided by /DIVISOR/.

List of procedures

TO SAW /COUNTER/ AND /LIMIT/
TO CHOOSE /COUNTER/ AND /LIMIT/
TO REM /NUMBER/ AND /DIVISOR/
TO PLOT /WORD/ AND /LENGTH/
TO REPEAT /WORD/ AND /LENGTH/

FLOW CHART FOR SAW


FIND THE REMAINDER OF /COUNTER/ DIVIDED BY /LIMIT/
IS /REMAINDER/ 0
YES
NO
PRINT REPEAT OF "X" AND /LIMIT/
PRINT PLOT OF "X" AND /REMAINDER/
ADD 1 TO /COUNTER/ TO BUILD /NEWCOUNTER/

ASSIGNMENT REPEAT

NAME OF PROCEDURE: TO REPEAT /WORD/ AND /LIMIT/

INPUTS: /WORD/ - any word or letter - must not be empty
/LIMIT/ - length of the new word - must be a number.
New word must be longer than /WORD/.

EXAMPLES:

```
←PRINT REPEAT OF "X" AND 2
XX
←PRINT REPEAT OF "APE" AND 6
AAAAPE
←PRINT REPEAT OF "X" AND 15
XXXXXXXXXXXXXXX
```

```
←LIST REPEAT
TO REPEAT /WORD/ AND /LIMIT/
10 CALL
     THING:  COUNT OF /WORD/
     NAME:  "TEST"
20 IS /TEST/ /LIMIT/
30 IF YES RETURN /WORD/
40 CALL
     THING:  FIRST OF /WORD/
     NAME:  "LETTER"
50 CALL
     THING:  WORD OF /LETTER/ AND /WORD/
     NAME:  "NEWWORD"
60 RETURN REPEAT OF /NEWWORD/ AND /LIMIT/
END
←
```

ASSIGNMENT PLOT

NAME OF PROCEDURE: TO PLOT /WORD/ AND /LIMIT/

INPUTS: /WORD/ - any word or letter
/LIMIT/ - length of the new word, including blanks.

EXAMPLES:

```
←PRINT PLOT OF "X" AND 9
        X
←PRINT PLOT OF "WHO" AND 4
 WHO
←PRINT PLOT OF "M" AND 1
M
←PRINT PLOT OF "M" AND 3
  M
←
```

REMINDER:

/BLANK/ is the blank letter.

```
←CALL
   THING:  WORD OF /BLANK/ AND "Y"
   NAME:  "SAMPLE"
←PRINT /SAMPLE/
 Y
←
```

```
←LIST PLOT
TO PLOT /WORD/ AND /LIMIT/
10 IS /LIMIT/ COUNT OF  /WORD/
20 IF YES RETURN /WORD/
30 CALL
      THING:  WORD OF /BLANK/ AND /WORD/
      NAME:  "NEWWORD"
40 RETURN PLOT OF /NEWWORD/ AND /LIMIT/
END
←
```

ASSIGNMENT REM

NAME OF PROCEDURE: TO REM /NUMBER/ AND /DIVISOR/

INPUTS: /NUMBER/ - any positive integer
/DIVISOR/ - any positive integer except 0.

OUTPUT: Remainder of /NUMBER/ divided by /DIVISOR/

EXAMPLES:

```
←PRINT REM OF 6 AND 3
0
←PRINT REM OF 3 AND 6
3
←PRINT REM OF 15 AND 2
1

←LIST REM

TO REM /NUMBER/ AND /DIVISOR/

10 IS /NUMBER/ GREATER OF /NUMBER/ AND /DIVISOR/

20 IF NO RETURN /NUMBER/

30 CALL
     THING:  DIFFERENCE OF /NUMBER/ AND /DIVISOR/

     NAME:  "NEWNUMBER"

40 RETURN REM OF /NEWNUMBER/AND /DIVISOR/

END

←
```

ASSIGNMENT CHOOSE AND SAW

NAME OF PROCEDURE: TO CHOOSE /COUNTER/ AND /LIMIT/

INPUTS: /COUNTER/ - any integer
/LIMIT/ - maximum word size - integer

EXAMPLES:

```
←PRINT CHOOSE OF 8 AND 4
XXXX
←PRINT CHOOSE OF 8 AND 5
  X

←LIST CHOOSE
TO CHOOSE /COUNTER/ AND /LIMIT/
10 CALL
     THING:  REM OF /COUNTER/ AND /LIMIT/
     NAME:  "REMAINDER"
20 IS /REMAINDER/  "0"
30 IF YES RETURN  REPEAT  OF "X" AND /LIMIT/
40 IF NO RETURN  PLOT  OF "X" AND /REMAINDER/
END

←LIST SAW
TO SAW /COUNTER/ AND /LIMIT/
10 PRINT CHOOSE OF /COUNTER/ AND /LIMIT/
20 CALL
     THING:  SUM OF /COUNTER/ AND "1"
     NAME:  "NEWCOUNTER"
30 RETURN SAW OF /NEWCOUNTER/ AND /LIMIT/
END
←
```

Guessing and Strategy

The next units introduced the students to search procedures and planning in the context of some simple mathematical games. The first of these games is, simply, to guess a number. After working with a procedure for blind guessing, the students were assigned work on a sequence of programs for guessing by binary partitioning (the binary search algorithm).

The games often make use of a special LOGO operation, /RANDOM/, which produces a digit at random whenever it is called. Thus,

```
←PRINT /RANDOM/
7
←PRINT /RANDOM/
4
←PRINT /RANDOM/
8
←PRINT WORD OF /RANDOM/ AND /RANDOM/
27
```

The assignment on blind guessing follows.

ASSIGNMENT RANDOMGUESS

Complete RANDOMGUESS and its associated procedure GUESS. Then use it to play several games. Record the number of guesses required before the computer guesses your number in each game.

```
TO RANDOMGUESS
10 PRINT "THINK OF A NUMBER BETWEEN 0 AND 99.  I'LL TRY TO
   GUESS IT."
20 GUESS
END

TO GUESS
10 CALL
     THING:  WORD OF /RANDOM/ AND /RANDOM/
     NAME:  "GUESS"
20 PRINT SENTENCE OF "MY GUESS IS" AND /GUESS/
30 PRINT "AM I RIGHT?  TYPE YES OR NO."
40 REQUEST "ANSWER"
50 IS /ANSWER/ "YES"
60 IF NO PRINT "I'LL TRY AGAIN."
70 IF NO GUESS
80 IF YES PRINT "LET'S PLAY ANOTHER GAME"
90 IF YES RANDOM GUESS
END
```

ASSIGNMENT NUMBERGUESS

RANDOMGUESS usually requires many trials before it guesses your number. We are going to develop a procedure NUMGUESS that will guess numbers a lot faster (i.e., with fewer guesses) than RANDOMGUESS (most of the time, not always - RANDOMGUESS is sometimes lucky).

The procedure NUMGUESS states the rules of the game to a beginning player and then calls the procedure NUMGAME. (Note that a player must say whether his number is HI or LO or OK - not just right or wrong as in RANDOMGUESS.) NUMGAME asks the player for some range of numbers that contains the number it is supposed to guess. (This interval need not be 0 to 99 as in RANDOMGUESS.) It then calls the procedure TRY (which does the real work). When TRY completes its job, by guessing the number, NUMGAME gives the player a chance to play another game.

```
TO NUMGUESS
10 PRINT "HI, DO YOU KNOW HOW TO PLAY THIS NUMBER GUESSING GAME?"
20 REQUEST "ANS"
30 IS /ANS/ "YES"
40 IF NO PRINT "YOU THINK OF A NUMBER, WHICH I WILL TRY TO GUESS.
   YOU HAVE TO GIVE ME SOME CLUES.  I NEED TO KNOW THE HIGHEST AS
   WELL AS THE LOWEST POSSIBLE NUMBERS WHICH YOU MIGHT CHOOSE."
50 IF NO PRINT /SKIP/        (/SKIP/ denotes a carriage return)
60 IF NO PRINT "AFTER I MAKE A GUESS, YOU MUST TELL ME IF THE
   GUESS IS HI, LO, OR OK (CORRECT)."
70 IF NO PRINT /SKIP/
80 RETURN NUMGAME
END
```

```
TO NUMGAME
10 TYPE "THE HIGHEST NUMBER IS "
20 REQUEST "TOP"
30 TYPE "THE LOWEST NUMBER IS "
40 REQUEST "BOTTOM"
50 TRY SENTENCE OF /TOP/ AND /BOTTOM/
60 PRINT /SKIP/
70 PRINT "DO YOU WANT TO PLAY AGAIN?"
80 REQUEST "ANS"
90 IS /ANS/ "YES"
100 IF YES RETURN NUMGAME
110 PRINT "GOODBYE"
END
```

Note that the procedure TRY has a single input - the <u>sentence</u> formed from the two numbers /TOP/ and /BOTTOM/ that specify the interval over which TRY is to guess. This interval is denoted /GAP/.

```
TO TRY /GAP/
10 CALL
     THING:  MIDDLE OF /GAP/
     NAME:  "GUESS"
20 PRINT SENTENCE OF "MY GUESS IS" AND /GUESS/
30 PRINT "AM I HI, LO, OR OK?"
40 REQUEST "ANSWER"
50 IS /ANSWER/ "HI"
60 IF YES RETURN TRY OF DOWN OF /GAP/ AND /GUESS/
70 IS /ANSWER/ "LO"
80 IF YES RETURN TRY OF UP OF /GAP/ AND /GUESS/
90 IS /ANSWER/ "OK"
100 IF YES PRINT "I GUESSED IT!"
110 IF YES RETURN
120 RETURN TRY OF /GAP/        (TRY is repeated if the answer is
END                             none of "HI", "LO", or "OK")
```

So TRY doesn't do all the real work either. It partitions its job among three other procedures - MIDDLE, DOWN, and UP. MIDDLE is used to compute a better guess, /GUESS/. If this guess is high, DOWN uses it to compute a tighter /GAP/ for TRY by decreasing the upper bound of the guessing interval; if this guess is low, UP uses it to compute a tighter /GAP/ for TRY by increasing the lower bound of the guessing interval.

MIDDLE is the key procedure in the program - it expresses the main idea of the binary search algorithm.

ASSIGNMENT MIDDLE

NAME OF PROCEDURE: TO MIDDLE /GAP/

INPUT: /GAP/ - a sentence of two integers. (The first integer is the upper bound of the search interval; the second integer is the lower bound.)

OUTPUT: The improved guess, the integer nearest to the midpoint of the search interval.
(MIDDLE uses a procedure which divides a number by 2. The answer is a two-word sentence: first the quotient, then the remainder. This procedure is discussed in the section on Arithmetic Procedures.)

EXAMPLES:

```
←PRINT MIDDLE OF "99 1"
50
←PRINT MIDDLE OF "100 75"
87
←PRINT MIDDLE OF "25 1"
13
←PRINT MIDDLE OF "11 10"
10
```

```
TO MIDDLE /GAP/
10 CALL
     THING:  SUM OF FIRST OF /GAP/ AND LAST OF /GAP/
     NAME:  "FIELD"
20 RETURN FIRST OF DIV2 OF /FIELD/
END
```

ASSIGNMENT UP and DOWN

NAME OF PROCEDURE: TO UP /GAP/ AND /GUESS/

INPUTS: /GAP/ - a two-word sentence

/GUESS/ - a word

OUTPUT: A two-word sentence (the reduced /GAP/) with /GUESS/ +1 as the improved lower bound.

EXAMPLES:

```
←PRINT UP OF "100 2" AND "50"
100 51
←PRINT UP OF "76 3" AND "10"
76 11
←
```

```
TO UP /GAP/ AND /GUESS/
10 RETURN SENTENCE OF FIRST OF /GAP/ AND SUM OF /GUESS/
   AND 1
END
```

- - - - - - - - - -

NAME OF PROCEDURE: TO DOWN /GAP/ AND /GUESS/

INPUTS: /GAP/ and /GUESS/ as in UP

OUTPUT: A two-word sentence (the reduced /GAP/) with /GUESS/ -1 as the improved upper bound.

EXAMPLES:

```
←PRINT DOWN OF "100 2" AND "50"
49 2
←PRINT DOWN OF "76 3" AND "10"
9 3
←
```

```
TO DOWN /GAP/ AND /GUESS/
10 RETURN SENTENCE OF DIFFERENCE OF /GUESS/ AND 1
   AND LAST OF /GAP/
END
```

NIM

The goal of this unit is to write a program for playing the game of NIM. The two players are the computer itself and a person (named "YOU"). NIM is a somewhat more complex mathematical game than number guessing but it has simple rules of play. The two players start with a common pile of chips. Taking turns, each player removes 1, 2, or 3 chips from the pile. The player who removes the last chip loses.

I. Plan of NIMPLAY

NIMPLAY requires two inputs - the number of chips currently in the game and the name of the next player.

NIMPLAY checks whether the game is completed. If it is not, it calls upon the next player to make his move.

TO NIMPLAY /CHIPS/ AND /PLAYER/

INPUTS:	/CHIPS/	/PLAYER/
	Number of Chips	Player: "COMP" or "YOU"

CHECK: IS GAME OVER?

ACTION: If it is, announce winner and done.
If it is not, /PLAYER/ moves.

PREPARATION FOR NEXT ROUND:	/NEWCHIPS/	/NEXTPLAYER/
	New number of chips	Next player: "YOU" or "COMP"

II. DETAILED SUBPLANS

A. CHECK

How do we check?

If /CHIPS/ is 1, the game is over. /PLAYER/ is the loser because he must take the last chip. In LOGO, we'll say

```
IS /CHIPS/ 1
IF YES PRINT SENTENCE OF /PLAYER/ AND "LOST!!"
```

If /CHIPS/ is 0, we also know the game is over. This time the previous player loses and /PLAYER/ wins. In LOGO, we write

```
IS /CHIPS/ 0
IF YES PRINT SENTENCE OF /PLAYER/ AND "WINS!"
```

B. PREPARATION

1. SUBPLAN FOR SETTING UP /NEWPLAYER/

This is easy. If /PLAYER/ is "YOU", /NEXTPLAYER/ is "COMP". If /PLAYER/ is "COMP", /NEXTPLAYER/ is "YOU". In LOGO, we say

```
IS /PLAYER/ "YOU"
IF YES CALL
      THING: "COMP"
      NAME: "NEXTPLAYER"
IF NO CALL
      THING: "YOU"
      NAME: "NEXTPLAYER"
```

2. SUBPLAN FOR SETTING UP /NEWCHIPS/

Since this problem is a little harder, we divide it into two SUBSUBPLANS. We will write two different procedures called: TO YOURPLAY /CHIPS/ and TO COMPLAY /CHIPS/.

Each procedure will change the number of chips. YOURPLAY can be pretty dumb. It has only to REQUEST a move. COMPLAY must be pretty smart. It has to figure out the best move.

ASSIGNMENT YOURPLAY

This procedure has the job of asking the human player (called "YOU") to choose 1, 2, or 3 and to return the number of chips left in the game.

We call the number of chips "CHIPS" and the number chosen by the player "MOVE".

Now look at the skeleton procedure on the next page. The real work is done by lines 20 and 90!

First make sure you understand these two lines. All the rest is to make sure YOU does not give a funny answer.

1. Fill in the following skeleton procedure.

```
TO YOURPLAY /CHIPS/
10 PRINT "YOU MAY TAKE 1, 2, OR 3."
20 REQUEST "MOVE"
30 IS /MOVE/ 1
40 IF NO IS /MOVE/ 2
50 IF NO IS /MOVE/ 3
60 IF NO RETURN YOURPLAY OF /CHIPS/
70 IS /CHIPS/ GREATER OF /CHIPS/ AND /MOVE/
80 IF NO RETURN YOURPLAY OF /CHIPS/
90 RETURN DIFFERENCE OF /CHIPS/ AND /MOVE/
END
```

2. Fill in the blanks in the following dialogs. Write what you think your procedure should do.

```
←PRINT YOURPLAY OF 7
YOU MAY TAKE 1, 2, OR 3.
*3
4
```

```
←PRINT YOURPLAY OF 4
YOU MAY TAKE 1, 2, OR 3.
*4
YOU MAY TAKE 1, 2, OR 3.
*2
2
```

```
←PRINT YOURPLAY OF 8
YOU MAY TAKE 1, 2, OR 3.
*1
7
```

```
←PRINT YOURPLAY OF 2
YOU MAY TAKE 1, 2, OR 3.
*3
YOU MAY TAKE 1, 2, OR 3.
* 1
1
```

3. Try your procedure at the terminal in the next class. Bring your copy of this sheet. Test whether your procedure does what you expected.

- - - - - - - - -

How should the computer decide its moves? Before we can write COMPLAY, we should devise a way to figure out best moves. Let's first consider easy cases with small numbers of chips.

If there is only one chip left, the computer loses. If there are 2 chips, the computer can take one thereby leaving one so it wins. Similarly, it wins for 3 or 4 chips by taking 2 or 3 and again leaving one. So 1 means lost, 2 means take 1, 3 means take 2, 4 means take 3. Now what about 5? If the computer takes 1, 2, or 3, the other player is left with 4, 3, or 2 and so can win if he plays correctly. So if the computer has 5 chips, it doesn't matter what it plays, it has to rely on a mistake by the other player in order to win. Five, like 1, is a bad number of chips to get.

Now, while 5 is bad for the computer, it is even worse for the opponent because the computer isn't going to make any mistakes in playing. So, if it gets 6, 7, or 8 chips, it will take 1, 2, or 3 leaving 5 and the opponent will lose. Thus, 6, 7, or 8 chips win for the computer.

If there are 9 chips left, no matter what the computer does, its opponent will be left in a favorable position (with either 6, 7, or 8 chips, all good numbers). So, 9 is bad. In the table we see -

1	lost	5	lost	9	lost
2	take 1	6	take 1	10	take 1
3	take 2	7	take 2	⋮	
4	take 3	8	take 3		

Of course, 5 and 9 are losing positions only if the opponent doesn't make a mistake. Hence, the computer won't give up even in those cases.

Notice the pattern in the table. For example, 4, 8, 12, --- all say take 3; 3, 7, 11, --- all say take 2; etc. Thus, to decide how many chips to take, the computer only needs to find out which of the four number sequences includes the current number of chips. We can now write a procedure that employs a strategy based on that observation.

TO COMPLAY /CHIPS/

Here is a strategy for playing NIM.

First divide the number of chips by 4 and find the remainder. For example,

REM OF 5 is 1
REM OF 27 is 3
REM OF 30 is 2
REM OF 83 is 3
REM OF 3 is 3

The rule for choosing a move is:

Remainder	Move
0	3
1	It doesn't matter. Say 3.
2	1
3	2

Apply this rule in the following cases:

/CHIPS/	REM OF /CHIPS/	/MOVE/	/NEWCHIPS/
10	2	1	9
13	1	3	10
15	3	2	13
11	3	2	9
17	1	3	14
12	0	3	9

Assume that we have a procedure LOOKUP which does this table look-up for us. Using it, we can make a procedure to choose the number of chips the computer will take on any move.

ASSIGNMENT CHOOSE

NAME OF PROCEDURE: TO CHOOSE /CHIPS/
INPUT: /CHIPS/ - The number of chips in the game.

EXAMPLES:

```
←PRINT CHOOSE OF 2
1
←PRINT CHOOSE OF 3
2
←PRINT CHOOSE OF 13
3
←
```

```
TO CHOOSE /CHIPS/
10 CALL
     THING:  REM OF /CHIPS/
     NAME:   "REM"
20 CALL
     THING:  LOOKUP OF /REM/
     NAME:   "MOVE"
30 RETURN /MOVE/
END
```

ASSIGNMENT REM

The NIM strategy needs to find the remainder of a number when divided by 4. One way to find the remainder is to keep subtracting 4.

```
Example 1:  /NUMBER/ is 17
            Subtract 4 from 17
            /NUMBER/ is 17 - 4 = 13
            Subtract 4 from 13
            /NUMBER/ is 13 - 4 = 9
            Continue until:
     STOP RULE:  /NUMBER/ isn't GREATER OF /NUMBER/ AND 4
     RETURN /NUMBER/
     17, 13, 9, 5, 1                    REM 17 = 1

Example 2:  /NUMBER/ is 18
     18, 14, 10, 6, 2                   REM  2 = 2

Example 3:  /NUMBER/ is 19
     19, 15, 11, 7, 3                   REM 19 = 3
```

Here is a program skeleton for this procedure in proper LOGO:

```
TO REM /NUMBER/
10 IS /NUMBER/ GREATER OF /NUMBER/ AND 4
20 IF NO RETURN /NUMBER/
30 CALL
     THING:  DIFFERENCE OF /NUMBER/ AND 4
     NAME:  "NEWNUMBER"
40 RETURN REM /NEW NUMBER /
END
```

What would the machine print if told -

```
TRACE REM
REM 11
REM "11"
   REM "7"
      REM "3"
      REM RETURNS "3"
   REM RETURNS "3"
REM RETURNS "3"
```

Fill in the result of

```
PRINT REM 13
1
```

```
PRINT REM 1789423
3 (after a long time)
```

```
PRINT REM "FOO"

GREATER OF "FOO" AND "4"? INPUTS MUST BE
NUMBERS.
I WAS AT LINE 10 IN REM.
```

ASSIGNMENT LOOKUP

We have to give CHOOSE a table look-up procedure.

NAME OF PROCEDURE: TO LOOKUP /NUMBER/
INPUT: /NUMBER/ is the answer received from REM OF /CHIPS/
This answer will be either 0, 1, 2, or 3.
OUTPUT: The number of chips the computer will take.

EXAMPLES:

```
←PRINT LOOKUP OF 3
2

←PRINT LOOKUP OF 2
1

←PRINT LOOKUP OF 1
3

←PRINT LOOKUP OF Ø
3
```

Write a procedure LOOKUP and test it.

ASSIGNMENT COMPLAY

We have written all the subprocedures needed for COMPLAY.

NAME OF PROCEDURE: TO COMPLAY /CHIPS/

INPUT: /CHIPS/ - The number of chips in the game.

EXAMPLES:

```
←PRINT COMPLAY OF 13
THE COMPUTER TAKES 3  }
10                    }       The machine's reply
←                     }

←PRINT COMPLAY OF 15
THE COMPUTER TAKES 2
13
←
```

```
TO COMPLAY /CHIPS/
10 CALL
     THING:  CHOOSE OF /CHIPS/
     NAME:  "MOVE"
20 PRINT SENTENCE OF "THE COMPUTER TAKES" AND /MOVE/
30 CALL
     THING:  DIFFERENCE OF /CHIPS/ AND /MOVE/
     NAME:  "NEWCHIPS"
40 RETURN /NEW CHIPS/
END
```

ASSIGNMENT NIMPLAY

We can now write the procedure which controls play.

NAME OF PROCEDURE: TO NIMPLAY /CHIPS/ AND /PLAYER/

INPUTS: /CHIPS/ - Number of chips in game
/PLAYER/ - Name of next player

EXAMPLE:

```
←NIMPLAY OF 17 AND "YOU"
YOU MAY TAKE 1, 2, OR 3
*
(IF YOU TYPE 2, IT SHOULD SAY)
15 CHIPS REMAIN
THE COMPUTER TAKES 2
13 CHIPS REMAIN
YOU MAY TAKE 1, 2, OR 3
*
AND SO ON.
```

```
←LIST NIMPLAY
TO NIMPLAY /CHIPS/ AND /PLAYER/
10 IS /CHIPS/ "0"
20 IF YES PRINT SENTENCE OF /PLAYER/ AND "WON!"
30 IF YES RETURN
40 IS /CHIPS/ "1"
50 IF YES PRINT SENTENCE OF /PLAYER/ AND "LOST!"
60 IF YES RETURN
65 PRINT SENTENCE OF /CHIPS/ AND "CHIPS REMAIN"
```

(continued)

```
70 IS /PLAYER/ "YOU"
80 IF YES CALL
     THING:  YOURPLAY OF /CHIPS/
     NAME:  "NEWCHIPS"
90 IF YES CALL
     THING:  "COMPUTER"
     NAME:  "NEWPLAYER"
100 IF NO CALL
     THING:  COMPLAY OF /CHIPS/
     NAME:  "NEWCHIPS"
110 IF NO CALL
     THING:  "YOU"
     NAME:  "NEWPLAYER"
120 RETURN NIMPLAY OF /NEWCHIPS/ AND /NEWPLAYER/
END
```

ASSIGNMENT SUPERNIM

Last of all, we write a procedure to set up a game.

NAME OF PROCEDURE: TO SUPERNIM

INPUTS: NONE

EXAMPLE:

```
←SUPERNIM
HOW MANY CHIPS DO YOU WANT IN THE GAME?
*            (The player must type a number)
DO YOU WANT TO GO FIRST?
*            (Player types YES or NO)
```

(SUPERNIM starts up NIMPLAY and tells it the number of chips in the game and the name of the first player)

(When NIMPLAY is finished, SUPERNIM continues its questions)

```
DO YOU WANT TO PLAY AGAIN?
*            (The player types YES or NO)
```

(If the answer is YES - SUPERNIM starts a new round and asks for the number of chips, etc.)

(If the answer is NO - SUPERNIM says BYE and stops)

Write a procedure SUPERNIM and then play some games with the computer.

Arithmetic Operations

LOGO does not have built-in multiplication or division operations -- these have to be written as procedures in terms of addition and subtraction. We gave students the assignment of writing their own integer multiplication and division procedures (though we provided fairly complete skeletons) to better understand how these operations work and can be used. At the same time, we did not think that the class would find arithmetic very appealing. We did not present multiplication and division procedures for their own interest but introduced them, rather, when they were needed by the students to do something else that was important to them (e.g., the pattern drawing and number guessing programs). To our surprise, however, most students enjoyed doing long multiplication and long division in the form of programs.

ASSIGNMENT MULTIPLY

LOGO has built-in procedures for adding (SUM) and subtracting (DIFF). Often, however, we have problems where the operations of multiplication and division are useful also. These aren't built into LOGO so we'll have to write them ourselves. Since it is always a good practice to do one thing at a time, and usually best to do the simplest things first (since that way if the harder is too hard, we shall at least have done something), we'll start with multiplication.

Well, it is easy enough to start. We just decide on a name for the procedure (say MULTIPLY) and its two inputs (say /X/ and /Y/).

But, how should we instruct LOGO to multiply two numbers? What does the product of 6 and 3 mean? One way to write it is 6+6+6. That seems a good way to write the procedure, just using addition. We want to add up /X/ the number of times given by /Y/. A neat way to do this is to use recursion. Here is our procedure:

```
←TO MULTIPLY /X/ AND /Y/
>10 IS /Y/ "0"
>20 IF YES RETURN "0"
>30 RETURN SUM OF /X/ AND MULTIPLY OF /X/ AND (DIFF OF /Y/ AND 1)
>END
MULTIPLY DEFINED
←
```

In ordinary (infix) notation, Line 30 says: X*Y = X+Y*(Y-1).

Now that we've got this procedure, let's use it. First, test it out with some numbers. Then, try these exercises which give you a chance to use MULTIPLY in some other procedures.

(1) Write a procedure called SQUARE whose output is the square of its input.

(2) Write a procedure called CUBE that cubes its input.

(3) Write a procedure (and think up a name for it) that takes for an input a sentence like "34 X 12" and returns the product, 408.

(4) Write a procedure that takes a number and multiplies it by 10. Can you do this without using MULTIPLY?

(5) Write a procedure called POWER that raises its first input to the power given by its second input. For example:
POWER OF "2" AND "3" would be 2 cubed or 8, and
POWER OF "3" AND "4" would be 3^4 or 3x3x3x3 or 81.
(Hint: The principle behind this one is very much like the principle behind the MULTIPLY procedure.)

(6) Write a procedure CASH /QUARTERS/ AND /DIMES/ AND /NICKLES/ AND /PENNIES/ that takes the number of each kind of coin and returns the number of cents it all comes to. For example, CASH OF "2" AND "1" AND "4" AND "8" would be 88.

ASSIGNMENT MULT

By now you've probably noticed that MULTIPLY isn't as fast as you might like, especially for large numbers. We can go a long way toward correcting this problem by using the trick mentioned in exercise (4) of the MULTIPLY assignment. This is the same trick that you've been using for years, ever since you learned to do long multiplication. Set up the multiplication problem 234x547 on a piece of paper and work it out. Your work probably looks like this (unless one of us made a mistake).

```
   234
   547
  ----
  1638
  936
1170
------
127998
```

Here, instead of multiplying by 547, all at once we multiplied by 7, then by 4, and then by 5. Mathematically speaking, what we've done is used the distributive law and said that 234x(500+40+7)=234x500 + 234x40 + 234x7 = 117000+9360+1638 = 127998. For us, multiplying by 500 is not appreciably slower than multiplying by 5 but for MULTIPLY it certainly is. When MULTIPLY multiplies by 5 it counts down 5,4,3,2,1,0, but when it multiplies by 500 it counts down 500,499,498,497,496, ..., 2,1,0, nearly 100 times as much work. We saved ourselves all that work by the trick of not multiplying by 500 all at once. We first multiplied by 5 in the normal way and then multiplied by 100 in a clever way that took almost no time at all. (Can you name the law that says multiplying by 500 gives the same answer as first multiplying by 5 and then by 100? If you can't, there is a big hint in exercise (1) following.)

How can we write a LOGO procedure that will use this trick and so be able to multiply 234x547 quickly? Well, what we want the procedure (let's call it MULT for fast multiplication) to do is to multiply 234 by 7, add that to 234x4x10, and add that to 234x5x100.

```
←TO MULT /X/ AND /Y/
>1Ø IS /Y/ /EMPTY/
>2Ø IF YES RETURN "Ø"
>3Ø RETURN SUM OF MULTIPLY OF /X/ AND (LAST OF /Y/) AND MULT OF
    (WORD OF /X/ AND "Ø") AND BUTLAST OF /Y/
>END
←
```

This is a pretty complex looking procedure. Some of the following exercises will help you understand it.

(1) Why do these two computations give the same answer?
(a) Multiply a number by 500. (b) Multiply the number by 5 and then multiply that answer by 100 Hint: In (a), write X*500 as X*(5*100). In (b), write X times 5 times 100 as (X*5)*100.

(2) Write line 3Ø of MULT in ordinary form using "327" for /X/ and "438" for /Y/. Remember that MULTIPLY and MULT both mean multiply, so use an X for MULTIPLY and an * for MULT.

(3) Your answer to (2) should be 327x8 + 3270*43. MULT will now be called again with /X/ as 3270 and /Y/ as 43. How will line 30 come out this time?

(4) Substitute the answer to (3) into the answer from (2) to get 327x8 + 3270x3 + 32700*4. So, MULT gets called again. This time round we get 327x8 + 3270x3 + 32700x4 + 327000*/EMPTY/. Now MULT finally gets to use line 20 and can finish. Work out 327x438 on paper and try to point out the similarities and differences between the way you do it and the way MULT does it.

(5) Do a complete round analysis [as in exercises (2), (3), and (4)] for MULT OF "73" AND "84".

(6) The commutative law says that AxB=BxA. Yet there is a difference between MULTIPLY OF "32576" AND "3" and MULTIPLY OF "3" AND "32576" even though the answers are the same. What is this difference? Try it out on the computer to make sure.

(7) The remark in exercise (6) is also true about MULT except that the difference is very much less in this case. Can you explain why?

- - - - -

After this work on multiplication, writing a procedure for division should seem a great deal easier.

ASSIGNMENT DIVIDE

NAME OF PROCEDURE: TO DIVIDE /DIVIDEND/ AND /DIVISOR/

INPUTS: /DIVIDEND/ - Any numeral
/DIVISOR/ - Any numeral

ANSWER: A two-word sentence - the first word the quotient - the second (last) word is the remainder.

```
←PRINT DIVIDE OF 7 AND 8
0 7
←PRINT DIVIDE OF 12 AND 3
4 0
←PRINT DIVIDE OF 100 AND 50
2 0
←PRINT DIVIDE OF 33 AND 10
3 3
←
```

To write DIVIDE we use a subprocedure called DIV.

```
TO DIVIDE /DIVIDEND/ AND /DIVISOR/
10 RETURN DIV OF /DIVIDEND/ AND /DIVISOR/ AND "0"
END
```

DIV has 3 inputs:
TO DIV /DIVIDEND/ /DIVISOR/ /QUOTIENT/
On the first round /QUOTIENT/ will be 0. On each round /QUOTIENT/ will increase. When the procedure stops, /QUOTIENT/ will be the proper answer. This is how it will work.

DIV 8 3 0

1st Round:

/DIVIDEND/, = 8 /DIVISOR/, = 3 /QUOTIENT/ = 0

Subtract /DIVISOR/ from /DIVIDEND/
Add 1 to /QUOTIENT/.

- - - - - -

2nd Round:

/DIVIDEND/, = 5 /DIVISOR/, = 3 /QUOTIENT/ = 1

- - - - - -

3rd Round:

/DIVIDEND/, = 2 /DIVISOR/, = 3 /QUOTIENT/ = 2

This time we do not subtract 3 from /DIVIDEND/. As soon as /DIVIDEND/ is smaller than /DIVISOR/ we stop.
/QUOTIENT/ should be the quotient of 8 divided by 3. It is.

- - - - - -

Write a LOGO procedure to do this. Don't forget the checks:
Stop when /DIVIDEND/ is smaller than /DIVISOR/.

ASSIGNMENT DIV

NAME OF PROCEDURE: TO DIV /DIVIDEND/ AND /DIVISOR/ AND /QUOTIENT/

INPUTS: /DIVIDEND/ - the dividend - any numeral
/DIVISOR/ - the divisor - any numeral

ANSWER: A two-word sentence - FIRST OF SENTENCE is the quotient- LAST OF SENTENCE is the remainder.

EXAMPLES:

```
←PRINT DIV OF 12 AND 5 AND 0
2 2
←PRINT DIV OF 14 AND 7 AND 0
2 0
←PRINT DIV OF 23 AND 12 AND 0
1 11
←PRINT DIV OF 42 AND 11 AND 0
3 9
←
```

ASSIGNMENT DIV

SKELETON

```
TO DIV /DIVIDEND/ AND /DIVISOR/ AND /QUOTIENT/
10 IS /DIVIDEND/ GREATER OF /DIVISOR/ AND /DIVIDEND/
20 IF NO RETURN SENTENCE OF /QUOTIENT/ AND /DIVIDEND/
30 CALL
     THING: DIFFERENCE OF /DIVIDEND/ AND /DIVISOR/
     NAME:  "NEWNUM"
40 CALL
     THING: SUM OF /QUOTIENT/ AND 1
     NAME:  "NEWQUO"
50 RETURN DIV OF /NEWNUM/ AND /DIVISOR/ AND /NEWQUO/
END
```

- - - - -

A fast division procedure ("long division") was then presented in a way similar to the development of MULT from MULTIPLY. Some special division procedures (like DIV2, division by 2) also were written as the need for them arose in various projects (such as the number guessing games).

CLOCK ARITHMETIC

During the course, the students were introduced to clock arithmetic (remainder arithmetic, modular arithmetic) in a few contexts.

Clock arithmetic base 4 had been used in NIM. Clock arithmetic with various bases had been used in the work on oscillators. Clock arithmetic based on 7 was used in designing a procedure DATEGAME to calculate the day of the week on which a given past (future) date fell (will fall).

```
←DATEGAME
TYPE THE DATE
*2 13 1944
THAT WAS A SUNDAY
←
```

(The students liked being able to assert to their peers authoritatively that 6+1 could be equal to zero.) A general clock arithmetic procedure was written and used in an interactive addition quiz. The person taking the quiz supplied his own problems (his typing is underscored):

```
←CLOCKADD
LET'S ADD IN REMARITH.  CHOOSE A DIVISOR.
WHAT DIVISOR DO YOU WANT?
*6
LET'S ADD IN 6 MINUTE CLOCK ARITHMETIC.

TYPE A NUMBER
*4
TYPE ANOTHER NUMBER
*5
WHAT IS THE SUM OF 4 AND 5?
*2
WRONG.  THE ANSWER IS 3.

TYPE A NUMBER
*7
TYPE ANOTHER NUMBER
*6
WHAT IS THE SUM OF 7 AND 6?
*1
RIGHT.
  .
  .
  .
```

Further programs for arithmetic operations were necessary in work arising in the algebra sequence. One example, the extension of multiplication to signed integers, is discussed in the next section.

Algebra Teaching Sequence

The last weeks of the course were spent on a sequence of classroom and laboratory assignments and projects that led our seventh graders into work on algebraic equation generation and the construction of algebra teaching programs. The sequence began with a unit on random sentence generators. The first problem was to construct a simple, but surprisingly useful, procedure called MEMBER.

Consider the problem of constructing a formal algorithm for the following process: given a number N and a list L find the Nth member of L. This problem was presented to children in something like the following form:

(a) Preliminary Explanation

MEMBER is an operation with two inputs. Examples of input and output are:

MEMBER "ABC" 1 = "A"
MEMBER "ABC" 3 = "C"
The intention is:
MEMBER /SENTENCE/ /NUMBER/ = /NUMBER/th word of /SENTENCE/

Questions for discussion: What are proper inputs and what are funny inputs? [We use "funny" as a technical word for the very important concept illustrated here for /SENTENCE/ and /NUMBER/.]

Proper
"AB" 1
"ABCD" 4
etc.

Funny
"AB" 0
"AB" 10
"AB" "CAT"
"A" -3
etc.

What should we do about funny inputs? List alternative solutions:

"ABC" 10 could be:

(1) Undefined, in which case the computer will complain, e.g.,

```
PRINT MEMBER "ABC" 10
THERE IS NO SUCH WORD IN THE SENTENCE
```

(2) Defined, in which case there is some output. What could the output be? Suggestions included:

/EMPTY/

"C" [always the last letter of /SENTENCE/]

"A" [because 10=1 in 3-clock arithmetic, as the children said it]

Further discussion would usually be deferred to a later stage. But the class understood:

that MEMBER was specified for a certain domain of inputs,
that it can be extended to a larger domain,
that the extension could be done in different ways,
that some ways are neater than others, e.g., /EMPTY/ and "A" are neater than "C" which is neater than "B" (at any rate, as far as we can judge on the justification given),
that the purpose of MEMBER will often determine the choice among possible extensions and, if so, will override the consideration of mathematical taste expressed by "neater than".

(b) Planning the Procedure

An example of a heuristic plan is

(1) Find easy cases
(2) Reduce the hard cases to easy ones.

The class learned that these heuristic plans do not always work -

but the possession of a collection of plans enabled one to "*do something*" when faced with a problem instead of being forced to sit in a trance and hope for inspiration.

The easy case for MEMBER is

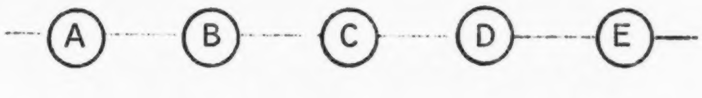
```
     /NUMBER/ = 1.
```

So we began by writing this part of the procedure:

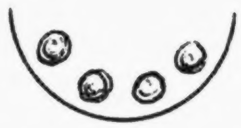
```
     TO MEMBER /SENTENCE/ /NUMBER/
     IS /NUMBER/ 1
     IF YES RETURN FIRST OF /SENTENCE/
```

Now we return to the *reduction* of the harder cases to easier cases. This idea was extensively discussed throughout the course together with heuristics for carrying out the reduction such as: set up a physical model. Although in this case a model would probably not have been necessary, we constructed one to illustrate the idea. In any case, if some children did not seem to be engaging their minds in the problem, we often urged them to invent a model as a constructive step.

Model for MEMBER "ABCDE" 4



Strings of beads representing
"A B C D E"



Bin of beads
representing 4

Question: How can one tell a child to find MEMBER "ABCDE" 4?
Answer: Take the beads out of the bin one at a time and peel beads off the string, one for one.

Discussion of the model led to:

The problem MEMBER /SENTENCE/ /NUMBER/

is equivalent to the problem

MEMBER BUTFIRST OF /SENTENCE/ DIFFERENCE OF /NUMBER/ AND 1

So we make two new things:

BF /SENTENCE/ DIFF /NUMBER/ 1	(where BF is the LOGO abbreviation for BUTFIRST, and DIFF for DIFFERENCE)

If we make new things, we should give them names; so let's use "NEWSENT" and "NEWNUM" as the new names.

(c) Procedures for MEMBER

```
←TO MEMBER /SENTENCE/ AND /NUMBER/
>10 IS /NUMBER/ 1
>20 IF YES RETURN FIRST OF /SENTENCE/
>30 CALL
    THING:  BUTFIRST OF /SENTENCE/
    NAME:  "NEWSENT"
>40 CALL
    THING:  DIFF OF /NUMBER/ AND 1
    NAME:  "NEWNUM"
>50 RETURN MEMBER OF /NEWSENT/ AND /NEWNUM/
>END
←
```

A shorter statement:

```
←TO MEMBER /S/ /N/
>10 IS /N/ 1
>20 IF YES RT F /S/
>30 RT MEMBER BF /S/ SUM /N/ -1
>END
←
```

(d) Adding Tests for Funny Inputs

```
>2 IS NUMBERP /N/ "TRUE"
>4 IF NO COMPLAIN
>6 IS ORDERP 1 /N/ COUNT /S/ "TRUE"
>8 IF NO COMPLAIN
```

These lines can be inserted in MEMBER to take care of certain kinds of funny inputs. They presuppose the procedures COMPLAIN and ORDERP:

```
←TO COMPLAIN
>10 PRINT SENTENCE OF SENTENCE OF "MEMBER IS NOT DEFINED FOR
    THE INPUTS" /S/ AND /N/
>END
←

←TO ORDERP /LOW/ /MIDDLE/ /HI/
>10 IS GREATERP /MIDDLE/ /HI/ "TRUE"
>20 IF YES RETURN "FALSE"
>30 IS GREATERP /LOW/ /MIDDLE/ "TRUE"
>40 IF YES RETURN "FALSE"
>50 RETURN "TRUE"
>END
←
```

Where NUMBERP is a predicate which has the output "TRUE" only if its input is a number; GREATERP is a predicate with two numerical inputs and whose output is "TRUE" only if its first input is greater than its second input.

Planning and debugging were learned through work with simple procedures such as MEMBER. Their real pay-off came in much more structured projects and teaching sequences. Thus, the final sequence of work assignments took our seventh grade children from MEMBER to making "random English" sentence generators of increasing complexity, then to algebraic equation generation and finally to writing algebra teaching programs.

(a) Random Sentence Generation

The following program, called RANDOMSELECT, selects a word randomly from a list.

```
←TO RANDOMSELECT /SENTENCE/
>10 CALL
     THING:  /RANDOM/
     NAME:  "NUMBER"
>20 RETURN MEMBER /SENTENCE/ /NUMBER/
>END
←
```

Note that RANDOMSELECT is simply a version of MEMBER that uses the operation /RANDOM/ for obtaining the second input.

A seventh grader's program, SIMPLESENTENCE, shown below, chooses at random a noun and a verb from two prescribed lists, /NOUNLIST/ and /VERBLIST/ which are LOGO sentences. It designates these "SUBJECT" and "ACTION", respectively. It then makes a sentence out of these and prints it. If /NOUNLIST/ contains words like "GIRLS" and "BOYS" and /VERBLIST/ contains words like "DANCE" and "FLY", SIMPLESENTENCE generates sentences like "GIRLS FLY" and "BOYS DANCE".

```
←TO SIMPLESENTENCE /NOUNLIST/ AND /VERBLIST/
>10 CALL
     THING:  RANDOMSELECT OF /NOUNLIST/
     NAME:  "SUBJECT"
>20 CALL
     THING:  RANDOMSELECT OF /VERBLIST/
     NAME:  "ACTION"
>30 PRINT SENTENCE OF /SUBJECT/ AND /ACTION/
>40 SIMPLESENTENCE /NOUNLIST/ AND /VERBLIST/
>END
←
```

The program is recursive - line 40 calls for the execution of SIMPLESENTENCE again and, since there is no STOP command, the program continues generating sentences endlessly.

SIMPLESENTENCE was the first of a series of programs constructed for generating even more elaborate grammatic English (and French) nonsense sentences. More complex programs were built upon the simpler ones. Thus, in quick succession, verbs were given an object, and adjectives and articles were incorporated. The extended SIMPLESENTENCE procedure was then used in compound sentence generators, like one that joined together simple sentences with connectors like "BECAUSE" and "WHILE".

The following is a sample of student printout from one of the later English sentence programs in the series.

```
THE MENTAL CAT DIGS THE WILD COMPUTER BECAUSE THE FUNNY BOY
LOVES THE CRAZY GIRL.

THE WILD DOG EATS THE GIRL ALTHOUGH THE BIG CAT CHASES THE
LOVELY COMPUTER WHILE THE GOOFY GIRL EATS THE WILD BOY.

A COMPUTER RUNS.
     : : :
```

In a brief excursion, appropriate (and, to these children, nontrivial) modifications were made in the English programs to make possible the generation of French sentences. Some results are illustrated by the following printouts. The programs were designed by the students; note the differences shown across these samples.

```
LE CAHIER EST GRAND
LA FILLE EST HAUTE
LE CAHIER EST HAUT
LE HORLOGE EST GRAND
LE CRAYON EST VERT
LA CHAISE EST PETITE
LA SERVIETTE EST NOIRE
LE SAC EST PETIT
LA GOMME EST GRANDE
LA MONTRE EST PETITE
LE CAHIER EST BRUN
LE SAC EST BRUN
LA GOMME EST VERTE
   .  .
   .  .
   .  .
```

```
MUR NOIR PRENDRE ET FILLE BLEU PARLER ET CRAYON BLEU PARLER
VERT ETROITE ENTRE ET ROBE PETTITE SORTIR ET CHIEN PETTITE FINIR
ROBE GRANDE PARLER ET CHAT ROSE ETRE ET VERT LARGE ALLER
SAC BLEU PRENDRE ET CHAT BLEU PRENDRE ET VERT ROSE FINIR
SAC GRIS PARLER ET VERT LARGE ETRE ET MUR PETTITE PRENDRE
   .   .
   .   .
   .   .
```

```
←FRENCH
WHEN YOU SEE THE FIRST * TYPE IN A NAME OF A FRENCH VERB (PLEASE DON'T
TYPE IN A ILLREGELAR VERB) WHEN YOU SEE THE SECOND * TYPE IN THE NOUN
YOU WANT ME TO WRITE
*PROMENER
*NOUS
NOUS PROMENONS
WHEN YOU SEE THE FIRST * TYPE IN A NAME OF A FRENCH VERB (PLEASE DON'T
TYPE IN A ILLREGELAR VERB) WHEN YOU SEE THE SECOND * TYPE IN THE NOUN
YOU WANT ME TO WRITE
*FINIR
*ELLES
ELLES FINISSENT
WHEN YOU SEE THE FIRST * TYPE IN A NAME OF A FRENCH VERB (PLEASE DON'T
TYPE IN A ILLREGELAR VERB) WHEN YOU SEE THE SECOND * TYPE IN THE NOUN
YOU WANT ME TO WRITE
*VENDRE
*JE
JE VENDS
    .   .
    .   .
    .   .
```

Algebra Quiz Programs

When the topic of sentence generation was introduced, students asked if that wasn't English rather than mathematics. The issue was resolved by the work in the next units, which were clearly about mathematics - arithmetic and algebra - yet, equally clearly derived from the earlier work on sentence generation.

The first unit, on algebra quiz programs, started with the observation that very slight modification of sentence generation programs allows one to generate mathematical sentences like those encountered in arithmetic. Thus, expressions like

```
"1 + 1 = 2"
"ONE PLUS ONE EQUALS TWO"
"2 + 2 = 3"
"A COMPUTER CAN TALK BUT IT CAN DO SUMS ALSO"
```

all are sentences in LOGO.

The first assignment was to make up an addition quiz program following the example shown in the procedures GENSUM and QUIZZ1 (see next page). Interestingly enough, the childrens' own programs incorporated considerably more English embellishment and conversation than ours. Samples of printouts from two of their programs are reproduced just after the assignment sheet.

ASSIGNMENT GENSUM

Here are two skeletal procedures which, together, make up a quiz program. QUIZZ1, the top-level procedure, asks whether a sentence like '5 + 3 = 8' is true or false. GENSUM generates the true/false sentence.

```
TO GENSUM
10 CALL
     THING:  /RANDOM/
     NAME:  "N1"
20 CALL
     THING:  /RANDOM/
     NAME:  "N2"
30 CALL
     THING:  SENTENCE OF SENTENCE OF /N1/ AND "+" AND /N2/
     NAME:  "LEFT"
40 IS GREATER OF /RANDOM/ AND "5" "5"
50 IF YES CALL
     THING:  SUM OF /N1/ AND /N2/
     NAME:  "N3"
60 IF NO CALL
     THING:  SUM OF /RANDOM/ AND /RANDOM/
     NAME:  "N3"
70 RETURN SENTENCE OF SENTENCE OF /LEFT/ AND "=" AND /N3/
END
```

```
TO QUIZZ1
10 PRINT SENTENCE OF SENTENCE OF "IS" AND GENSUM AND "TRUE OR
   FALSE?"
20 REQUEST "ANSWER"
30 PRINT SENTENCE OF "I THINK YOU REALLY MEANT TO SAY" AND
   /ANSWER/
40 PRINT "LET'S TRY ANOTHER"
50 RETURN QUIZZ1
END
```

Try out QUIZZ1 and then make up your own arithmetic quiz program.

```
←SCOOBA

COME ON WAKE UP I AM YOUR FRIENDLY ADDITION MAN I HOPE YOU KNOW
ADDITION DO YOU ?
*YES
0 + 5
*5

YOU ARE SMART BUT NOT BRILLANT BECAUSE YOU GOT IT RIGHT LETS TRY AGAIN

1 + 9
*10

YOU ARE SMART BUT NOT BRILLANT BECAUSE YOU GOT IT RIGHT LETS TRY AGAIN

8 + 6
*15

YOU IDIOT I THOUGHT YOU KNEW ADDITION LETS TRY AGAIN AND SEE IF YOU
KNOW IT THIS TIME
8 + 6
*14
YOU ARE SMART BUT NOT BRILLANT BECAUSE YOU GOT IT RIGHT LETS TRY AGAIN

         ⋮    ⋮


←SCOOBA

COME ON WAKE UP I AM YOUR FRIENDLY ADDITION MAN I HOPE YOU KNOW
ADDITION DO YOU ?
*NO
YOU DUMMY DONT YOU KNOW ADDITION YOU WERE SUPPOSE TO LEARN IT IN THE
FIRST GRADE IF YOU DONT KNOW IT HERE IS A SIMPLE PROBLEM WHAT IS THE
SUM OF 1 + 1 TO FIGURE THIS OUT I WILL DRAW 2 XS X X NOW COUNT THEM UP
AND WHAT IS THE SUM ?
*2
IF YOUR SO SMART WHAT IS THE SUM OF 5 + 3
*2
YOU SAID YOU KNEW ADDITION LETS TRY AGAGAIN
5 + 3
*8
YOUR VERY SMART BUT NOT AN EXPERT YET
IF YOUR SO SMART WHAT IS THE SUM OF 8 + 1

         ⋮    ⋮
```

```
-TALKMATH
HERE I AM A MATH PROFFESSOR.NEVER THOUGHT I'D MAKE IT,DID YOU? I'M
GOING TO TEST YOU AND SEE HOW SMART YOU ARE.
ARE YOU SMART OR STUPID?
*SMART
WELL A SMART STUDENT HOPE YOURE NOT JUST SAYING YOURE SMART ANYWAY ,TRY
THESE.
2 * 0 =
WHAT IS THE ANSWER?
*0
EXCELLENT.BRAVO! NICE WORK.NOW TRY SOME MORE
9 * 5 =
WHAT IS THE ANSWER?
*45
EXCELLENT.BRAVO! NICE WORK.NOW TRY SOME MORE
8 * 8 =
WHAT IS THE ANSWER?
*45
TOO BAD BUT YOU GOT THEM WRONG (YOU DUMMY).THE ANSWER IS 64
1 * 4 =
WHAT IS THE ANSWER?
        :    :
        :    :

-TALKMATH
HERE I AM A MATH PROFFESSOR.NEVER THOUGHT I'D MAKE IT,DID YOU? I'M
GOING TO TEST YOU AND SEE HOW SMART YOU ARE.
ARE YOU SMART OR STUPID?
*STUPID
SO YOU'RE STUPID.I'LL TRY TO GIVE YOU EASY PROBLEMS.
9 + 3 =
WHAT IS THE ANSWER?
*12
VERY GOOD,AND I THINK YOU'RE OFF TO A GOOD START.MAYBE WE CAN TRY
ANOTHER ONE.
4 + 7 =
WHAT IS THE ANSWER?
*11
VERY GOOD,AND I THINK YOU'RE OFF TO A GOOD START.MAYBE WE CAN TRY
ANOTHER ONE.
1 + 6 =
WHAT IS THE ANSWER?
*13
BOY!WHEN YOU SAY YOURE STUPID YOURE NOT KIDING! 7 THATS THE ANSWER.
1 + 9 =
WHAT IS THE ANSWER?
        :    :
        :    :
```

In both of the examples shown, the students chose to take the stance (and the tone, as they apparently see it) of a teacher, and a rather strict one. Note that in the second of these examples, the "smart" student is given multiplication problems instead of addition problems. In both cases the mathematics is carried out more correctly than the English. Both transcripts show a directness in word choice that was not mirrored in these students' language arts classes. (We neither encouraged nor discouraged them in their choice of words for praise or insult, since these never exceeded acceptable bounds. Our own programs obviously did not always serve as their models.)

Another algebra quiz assignment, TALKALGEBRA, concerning addition word problems with signed numbers, was very closely descended from English sentence generating procedures like SIMPLESENTENCE. TALKALGEBRA uses two procedures - CHOOSE and PICK - that are essentially the same as MEMBER and RANDOMSELECT. TALKALGEBRA uses a procedure ALGTALK to generate a random number of sentences such as

```
I GET 3 PIES
I LOSE 6 LOBSTERS
I BUY 2 TRUFFLES
```

where the numbers preceding the objects are chosen randomly, and then queries the user on how many things remain.

In writing their variants of TALKALGEBRA, the students chose their own words for the objects ("GOODIES"), the positive words like "GET", and the negative words like "LOSE". After the listing of the assignment, we show copies of two students' programs. Note that in the second sample, the student has incorporated negative numbers in his sentences.

ASSIGNMENT TALKALGEBRA

```
TO CHOOSE /SENTENCE/ AND /NUMBER/
10 IS /NUMBER/ "0"
20 IF YES RETURN FIRST OF /SENTENCE/
30 RETURN CHOOSE OF BUTFIRST OF /SENTENCE/ AND DIFFERENCE OF
   /NUMBER/ AND "1"
END


TO PICK /SENTENCE/
10 CALL
     THING:  /RANDOM/
     NAME:  "NUMBER"
20 IS /NUMBER/ GREATER OF /NUMBER/ AND COUNT OF /SENTENCE/
30 IF YES RETURN PICK OF /SENTENCE/
40 RETURN CHOOSE OF /SENTENCE/ AND /NUMBER/
END


TO TALKALGEBRA
10 CALL
     THING:  "PIES TRUFFLES LOBSTERS"
     NAME:  "GOODIES"
20 CALL
     THING:  "MAKE GET BUY FIND"
     NAME:  "POSITIVEWORDS"
30 CALL
     THING:  "LOSE SELL BREAK GIVE"
     NAME:  "NEGATIVEWORDS"
40 ALGTALK OF "0"
END


TO ALGTALK /TOTAL/
5 CALL
     THING:  PICK OF /GOODIES/
     NAME:  "OBJECT"
10 CALL
     THING:  /RANDOM/
     NAME:  "NUMBER"
20 IS PICK OF "+ -" "+"
30 IF YES CALL
     THING:  PICK OF /POSITIVEWORDS/
     NAME:  "ACTION"
40 IF YES CALL
     THING:  SUM OF /TOTAL/ AND /NUMBER/
     NAME:  "TOTAL"
```

(continued)

```
50 IF NO CALL
     THING:  PICK OF /NEGATIVEWORDS/
     NAME:  "ACTION"
60 IF NO CALL
     THING:  DIFFERENCE OF /TOTAL/ AND /NUMBER/
     NAME:  "TOTAL"
70 PRINT SENTENCE OF SENTENCE OF SENTENCE OF "I" AND /ACTION/
   AND /NUMBER/ AND /OBJECT/
80 IS PICK OF "STOP GO GO GO GO GO GO" "GO"
90 IF YES RETURN ALGTALK OF /TOTAL/
100 IF NO PRINT "HOW MANY THINGS DO I HAVE NOW?"
110 REQUEST "ANSWER"
120 IS /ANSWER/ /TOTAL/
130 IF YES PRINT "CALLOO CALLAY"
140 IF NO PRINT "THAT IS NOT SO"
150 RETURN ALGTALK OF "0"
END
```

Make up your own words for TALKALGEBRA. Modify the procedures some other ways you can think of, too.

- - - - - -

```
←TALKALGEBRA
I TAKE 3 EGGS
I MAKE 1 FISH
HOW MANY THINGS DO I HAVE NOW?
:4
CALLOO CALLAY
I GIVE 3 CAKES
I SELL 7 FISH
I GET 1 MARMALADE
I TAKE 3 CAKES
HOW MANY THINGS DO I HAVE NOW?
:3
THAT IS NOT SO
     :
```

```
←TALKALGEBRA
I LOSE 8 LOBSTERS
I STEAL 6 SUBS
I BREAK 5 PIES
HOW MANY THINGS DO I HAVE NOW?
*-7
BOY THATS NEAT IT TOOK ME ALONG TIME TO WORK OUT HOW THIS DUM PROCEDURE
WORKS, BUT LOOK AT YOU WIZZIN BY IT LIKE IT WAS JUST ANOTHER PROBLEM
I GET -1 LOBSTERS
HOW MANY THINGS DO I HAVE NOW?
*-4
TO BAD -1
I GET -0 PIES
I STEAL 9 LOBSTERS
I GET 4 LOBSTERS
I BOMB -5 PIES
I BOMB 0 LOBSTERS
I BUY -3 LOBSTERS
I MAKE 8 LOBSTERS
I BREAK -8 LOBSTERS
I BREAK 5 SUBS
I GET 3 LOBSTERS
I STEAL -9 PIES
I BOMB -2 SUBS
I BREAK 3 PIES
HOW MANY THINGS DO I HAVE NOW?
*19
BOY THATS NEAT IT TOOK ME ALONG TIME TO WORK OUT HOW THIS DUM PROCEDURE
WORKS, BUT LOOK AT YOU WIZZIN BY IT LIKE IT WAS JUST ANOTHER PROBLEM
     :   :
     :   :
```

Algebra Teaching Programs

The children already knew how to solve the addition problems generated in the two quiz programs just described. In the next unit, the children generated quiz problems that they didn't know how to solve, except by trial and error in the simplest instances. These new quiz problems were linear equations like those encountered in ninth-grade algebra:

$$7X + 8 = 71$$
$$2X + 3 = 9$$
$$\dots$$

In the childrens' programs, the coefficients were randomly chosen by RANDOMSELECT and the problems were stated as questions of the form:

```
     3 * /BOX/ + 5 = 11
     WHAT IS /BOX/?
```

The following assignment introduced this new unit.

TEACHING ALGEBRA

In this unit we shall make procedures for teaching 9th-grade algebra. One reason for doing this is the following theory: if you teach the computer to teach 9th graders how to do algebra, then maybe you will teach yourselves how to do it at the same time.

We shall use sentence generating and guessing games to help make the teaching program.

A sentence like

3 x ☐ + 4 = 10

is a simple EQUATION. Finding out what number to put in the box to make the sentence true is called SOLVING the equation.

In typing the equation for LOGO, we shall write it as

```
     3 * /BOX/ + 4 = 10
```

Notice carefully:

(1) We use * instead of x so as to avoid confusing "times" and the letter "X".

(2) We use /BOX/ because "BOX" is the name of the number we are going to find.

(3) The equation is a LOGO sentence, so it must be typed in with proper spaces. So 3*/BOX/ +4 =6 is wrong: 3*/BOX/ should be <u>three</u> <u>words</u>. 3 * /BOX/ + 4 = 6 is right; it has seven words.

To make sure you understand all this, try these exercises:

```
←CALL
   THING:  "3 * /X/ + 4 = 10"
   NAME:  "E"
```

What is:

(1) F /E/ 3

(2) L /E/ 10

(3) F BF /E/ *

(4) L BL BL /E/ 4

(5) F BF BF /E/ /x/

(6) W OF W OF F OF BF OF /E/ AND F OF BF OF BF OF BF OF /E/ AND F OF BF OF BF OF BF OF BF OF BF OF /E/ *+=

The equation, /E/, is simple enough for you to solve. Write down the THING of "X" that makes /E/ true.

/X/ = 2

- - - - - -

The program for generating equations operates as follows. A random number is chosen for the coefficient of /BOX/ (this is called "TIMESNUM"); a second random number is chosen for the last coefficient (it is called "SUMNUM"); then, instead of choosing a random number for the right side of the equation, as one might have expected, the last random number is chosen for /BOX/ itself.

(That is the trick that made it possible for the children to know the answer to the problem even though they themselves could not solve it.)

As an example, if the program picked 4 for /TIMESNUM/, 2 for /SUMNUM/, and 7 for /BOX/, it would compute 4 * 7 + 2 (using integer MULTIPLY procedures primarily written by the children) and then print:

```
4 * /BOX/ + 2 = 30
WHAT IS /BOX/?
```

The program would then wait for an answer to be typed in. The seventh graders were given the problem of deciding whether the answer was right or wrong and what to do in either case.

The following sample output was typical of their first attack on this problem.

Computer:

```
6 * /BOX/ + 9 = 27
WHAT IS /BOX/?
```

User:

```
2
```

Computer:

```
HA HA.  WRONG.
DONT YOU KNOW THE RIGHT ANSWER IS 3?
```

They soon incorporated frills of various kinds, like large coefficients, negative coefficients, the use of the CLOCK operation to measure how long the user took to answer, and so on. For example, the following assignment shows how MULTIPLY was changed to take negative as well as unsigned (positive) inputs. It uses the old MULT procedure which takes positive inputs.

ASSIGNMENT NEGATIVES

Purpose: to be able to use negative numbers in the algebra procedures.

(1) Change MULTIPLY so that it multiplies negative numbers. Use these procedures.

```
TO COUNTNEG /X/ AND /Y/
10 CALL "0" "COUNT"
20 IS FIRST OF /X/ "-"
30 IF YES CALL SUM OF /COUNT/ AND "1"   "COUNT"
40 IS FIRST OF /Y/ "-"
50 IF YES CALL
     THING:  SUM OF /COUNT/ AND "1"
     NAME: "COUNT"
60 RETURN /COUNT/
END

TO ABSOLUTE /X/
10  IS FIRST OF /X/ "-"
20 IF YES RETURN BUTFIRST OF /X/
30 RETURN /X/
END

TO MULTIPLY /X/ AND /Y/
10 CALL
     THING:  MULT OF ABSOLUTE OF /X/ AND ABSOLUTE OF /Y/
     NAME:  "PRODUCT"
20 IS COUNTNEG OF /X/ AND /Y/ "1"
30 IF YES RETURN WORD OF "-" AND /PRODUCT/
40 RETURN /PRODUCT/
END
```

(2) Now make the algebra program generate equations like

-3 * /X/ + 4 = 1

In discussing the issue of how they might modify their programs to help a user who was having difficulty solving a problem, the seventh graders began to think about how to solve these problems themselves. (Remember - because of the tricky way of generating the problems, our students knew the right answers even though they did not know an algorithm for solving these equations, nor indeed that there was such a thing as an algorithm for solving equations.)

One idea was to show when an answer was wrong that it plainly did not satisfy the equation. Thus,

Computer:

```
3 * /BOX/ + 5 = 17
WHAT IS /BOX/?
```

User:

```
2
```

Computer:

```
YOU ARE WRONG.  IF /BOX/ WAS 2,
3 * /BOX/ + 5 WOULD BE 11, NOT 17.
```

The following sample printout shows a student's variant of this in a mixed quiz incorporating different kinds of problems.

```
←ALGEBRA
TRY SOME OF THESE PROBLEMS :
3 + 7 = /E/
*10
YOU GOT IT
TRY SOME OF THESE PROBLEMS :
3 + 9 = /E/
*12
YOU GOT IT
TRY SOME OF THESE PROBLEMS :
2 * /BOX/ + 7 = 15
*4
YOU GOT IT BUT IT TOOK YOU 10 SECONDS
```

```
7 * /BOX/ + 2 = 30
*24
YOU GOT IT WRONG TRY IT AGAIN !
7 * 24 = 168
168 + 2 = 170
*14
7 * 14 = 98
98 + 2 = 100
*4
6 * /BOX/ + 4 = 10
*1
 YOU GOT IT BUT IT TOOK YOU 7 SECONDS
8 * /BOX/ + 0 = 0
*0
YOU GOT IT BUT IT TOOK YOU 3 SECONDS
5 * /BOX/ + 0 = 25
*5
YOU GOT IT BUT IT TOOK YOU 3 SECONDS
        .   .
        .   .
        .   .
```

Several children tried to do more. They realized that there might be an explicit way of telling the user how to solve the problem for any problem of this kind! By intense effort, often involving extensive trial-and-error, some of the children were successful in finding the algorithm! A sample printout from such a successful program follows.

```
←ALGE
-38 * /BOX/ + +28 = -124

WHAT IS /BOX/ ?
*4
IT TOOK YOU 21 SECONDS TO ANSWER ME YOU KNUCKELBRAIN THAT IS SLOW!
WRONG
THE REAL ANSWER IS +04
AN EASY WAY TO GET THE ANSWER IS TO SUBTRACT +28 FROM -124 AND THEN TRY
TO DIVIDE -38 INTO -152
```

```
-78 * /BOX/ + +97 = -3023

WHAT IS /BOX/ ?
*+35
IT TOOK YOU 33 SECONDS TO ANSWER ME YOU KNUCKELBRAIN THAT IS SLOW!
WRONG
THE REAL ANSWER IS +40
AN EASY WAY TO GET THE ANSWER IS TO SUBTRACT +97 FROM -3023 AND THEN
TRY TO DIVIDE -78 INTO -3120

-31 * /BOX/ + +50 = -2802

WHAT IS /BOX/ ?
*+92
IT TOOK YOU 19 SECONDS TO ANSWER ME YOU KNUCKELBRAIN THAT IS SLOW!

GOOD
-54 * /BOX/ + +09 = -477

WHAT IS /BOX/ ?
*2
YOU MUST BE A BRAIN TO ANSWER ME IN 5 SECONDS
WRONG
THE REAL ANSWER IS +09
AN EASY WAY TO GET THE ANSWER IS TO SUBTRACT +09 FROM -477 AND THEN
TRY TO DIVIDE -54 INTO -486
      .   .
      .   .
      .   .
```

In the example just shown, the student incorporated large negative numbers with a vengeance! Other students pursued different goals. For example, the following printout shows the amalgamation into a quiz of some personal, non-mathematical problems. The student did not go so far with algebra problems as the last one, but he started from a more remote and alien mathematical past and his relative progress was equally as impressive to us.

```
←MULTEACH
4 * /BOX/ = 20
WHAT IS /BOX/?
*5
VERY GOOD. YOUR A SMART LITTLE DEVIL. BUT YOU TOOK 3 SECONDS
WOULD YOU LIKE TO NO MORE
*YES
ITS BEEN LIKE THIS, YOU SEE ALONG TIME AGO MY MOTHER SAID I WAS NOT
AGING LIKE ALL THE OTHERS, SO WE GOT A CUPUTER DOCTOR TO HELP ME, AND
HE DID. SO NOW YOU NO WHY I AM SO OLD
8 * /BOX/ = 32
WHAT IS /BOX/?
*4
VERY GOOD. YOUR A SMART LITTLE DEVIL. BUT YOU TOOK 2 SECONDS
WOULD YOU LIKE TO NO MORE
*NO
BOY ITS NOT OFTEN YOU HERE A STORY LIKE MINE, BUT SINCE YOU DO NOT I
NOW RETURN YOU TO YOUR SO CALLED HUMAN FUN HA HA HA
8 * /BOX/ = 48
WHAT IS /BOX/?
*8
/BOX/ IS 6
IF YOU THINK YOUR SO SMART WHY DID YOU GET IT WRONG. OR WHERE YOU
THINKING OF TRICKING THEY OLDEST COMPUTER IN THE WORLD. PLEASE DO NOT I
AM 558 YEARS OLD AND DO NOT WISH TO DIE NOW.
WOULD YOU LIKE TO NO MORE
*NO
BOY ITS NOT OFTEN YOU HERE A STORY LIKE MINE

      ⋮    ⋮
```

At the end of the course, students were extending their teaching programs to include equations of variable form and we were beginning to incorporate algebra word problems into new teaching programs.

4.4 Evaluation

This section comprises the results and conclusions of the research described in the body of the report. The difficulty of evaluating limited educational experiments by objective measures is well known to us. Nevertheless, we have done some testing. We prefer the test of critical judgment by appropriate persons - intelligent, informed, and truly objective mathematicians and mathematics educators - based on direct personal contact with the children. We have carried out some evaluation of this kind also.

We first discuss the results of standard testing of our students with the Iowa Tests of Basic Skills. We then discuss phase (i.e., track) placement, a rank measure of achievement level that is standardly used in the Lexington school system. Since phase placement decisions about a student are made by one teacher but reconsidered by others, change of phase placement is a compound subjective measure of progress. We next include the judgments made by four experienced and well-known members of the mathematics-education community who each made several trips to the classroom to monitor the teaching experiment at first hand. Last of all, we give our own judgments.

Achievement Test Results

Each year, in October, all students at Muzzey Junior High School take the Iowa Tests of Basic Skills (ITBS), a standardized achievement test used in many schools to measure student performance as a guide to student placement. We planned to compare 1968 and 1969 ITBS results of our experimental class with those of comparable children at the school. We discussed this use of these tests and other evaluative procedures with the staff at Muzzey a number of times during the year. Messrs. Santo Marino, school Principal, David Terry, Assistant Principal, and Robert Patterson, Guidance Counselor, generously and helpfully consulted with us and made available the data we requested. The school volunteered to identify and select a matched control group of twelve students for comparison of ITBS scores and other measures with the twelve students in our experimental class.

We understand the limitations of such small samples. We had planned to use a larger control group, consisting of all seventh grade children at the same level of mathematical achievement as the experimental class. We would then have been comparing the performance of the experimental class with a more reliable control. In scoring the ITBS, the raw scores are usually converted, first to grade equivalents, and then to local or national norms. Because the ITBS is administered in October, there was not enough time to get the scores back from the testing service; in order to get any results, we had to score them by hand. Consequently, we could not, within the time requirements of this report, process even the raw scores for the students in the large control group we had planned to use. We thus had to be satisfied with the smaller control group chosen by the school and, as it is, we have available for study, raw scores for 1968 and 1969 for twenty-four children. The ITBS publisher, Houghton Mifflin Company, confirmed our assumption that raw scores are as unbiased a measure of comparison as converted scores.

The ITBS has eleven independent sections. These are called: Vocabulary, Reading, Spelling, Capitalization, Punctuation, Usage, Map Reading, Reading Graphs and Tables, Use of Reference Material, Arithmetic Concepts, and Arithmetic Problems. We computed averages for each section for each group for each year. In addition, we tallied the number of questions answered correctly by each student for each year.

Number of Correct Answers in ITBS

		Range for Individual Students	Grand Total
7th Grade (1968)	(Computer)	166 - 298	2896
	(Control)	214 - 371	3174
8th Grade (1969)	(Computer)	144 - 305	3010
	(Control)	209 - 382	3180

Although we have noted some trends, we cannot be sure how significant they are. We can only say that for the twelve children in the computer class and the twelve children chosen as a matched control group, some things are true of their raw scores:

(1) The control group has a much higher range of scores and a much higher grand total, thus suggesting that this group, by these standards, is not closely matched to the computer class.

(2) Both groups show a widening of the range from 7th to 8th grade, i.e., the lowest score is lower and the highest score is higher in the eighth grade.

(3) The changes in the control group are very small: down 5 on the low side, up 11 on the high, up 6 on the overall total. The computer class went down 22 on the low side, up 7 on the high, but up 114 on the total.

(4) In the change in individual totals, the control group was mixed: 5 student totals went down, 7 went up. The computer group totals show only two students going down, markedly (-22 and -25) and predictably from our class experience of their general outlook. All the other computer student totals went up.

(5) The average change in total score was +0.5 for the control group and +9.5 for the computer class.

These observations are based on individual test data shown in Tables I - III on the pages following.

Conclusions drawn from these data are subject to widely varying interpretation. We can hesitantly say that the computer class showed markedly positive changes, relative to the control group, in the sections on Vocabulary, Reading, Use of Reference Material, Reading Graphs and Tables, and Arithmetic Concepts. On the other hand, the control group did better than the computer class in Capitalization, Punctuation, Map Reading, and Arithmetic Problems. (The difference between the changes on Arithmetic Problem scores was not large and is possibly due to the fact that the computer class did not get much work with standard seventh-grade arithmetic problems during the year.)

We can confidently say that the achievement test results indicate that the computer class childrens' progress in mathematics and other subjects was not adversely affected by their experience.

TABLE I

COMPUTER CLASS

RAW SCORES — IOWA TESTS OF BASIC SKILLS

7th Grade – October 1968 8th Grade – October 1969
Muzzey Junior High School, Lexington, Massachusetts

Student Number		Vocabulary (48)*	Reading (80)	Spelling (48)	Capitalization (44)	Punctuation (44)	Usage (32)	Map Reading (42)	Reading Graphs and Tables (28)	Use of Reference Material (59)	Arithmetic Concepts (48)	Arithmetic Problems (34)	Grand Totals
1 –	7th Grade	33	45	29	26	28	24	15	14	26	32	13	285
	8th Grade	33	58	28	26	24	22	22	13	34	30	15	305
2 –	7th Grade	32	21	23	23	18	26	18	13	21	17	15	227
	8th Grade	28	38	29	8	17	23	19	14	24	21	8	229
3 –	7th Grade	19	31	16	24	26	14	13	8	14	15	8	188
	8th Grade	26	39	22	24	20	16	12	12	22	16	10	219
4 –	7th Grade	21	25	18	32	32	23	13	17	37	16	13	247
	8th Grade	20	43	24	33	35	19	26	12	25	24	16	277
5 –	7th Grade	15	38	27	31	29	18	21	15	34	28	11	267
	8th Grade	21	48	23	32	25	13	22	11	40	27	14	276
6 –	7th Grade	11	34	18	17	14	18	10	6	18	13	7	166
	8th Grade	18	23	16	9	9	6	12	9	15	19	8	144
7 –	7th Grade	20	28	26	31	21	16	14	14	26	31	13	240
	8th Grade	27	44	27	23	19	17	23	11	37	25	11	264
8 –	7th Grade	29	43	16	26	24	22	18	15	40	22	12	267
	8th Grade	20	53	25	27	25	20	21	17	43	23	10	284
9 –	7th Grade	28	38	33	17	15	19	11	17	33	15	21	247
	8th Grade	31	34	30	16	16	13	19	15	35	18	21	248
10 –	7th Grade	33	44	32	23	18	25	14	10	26	18	11	254
	8th Grade	26	28	38	27	22	16	13	17	16	20	6	229
11 –	7th Grade	33	49	27	14	14	12	11	7	19	20	4	210
	8th Grade	26	51	29	12	18	14	14	11	32	21	8	236
12 –	7th Grade	26	42	40	32	25	22	17	13	40	27	14	298
	8th Grade	31	48	38	31	22	24	20	7	43	20	15	299
AVERAGES	7th	25.0	36.5	25.4	24.7	22.0	19.9	14.6	12.4	27.8	21.2	11.8	241.3
	8th	25.6	42.3	27.4	22.3	21.0	16.9	18.6	12.4	30.5	22.0	11.8	250.8

---

*The number of items in the test sections.

TABLE II

CONTROL GROUP

RAW SCORES — IOWA TESTS OF BASIC SKILLS

7th Grade – October 1968 8th Grade – October 1969
Muzzey Junior High School, Lexington, Massachusetts

Student Number		Vocabulary (48)*	Reading (80)	Spelling (48)	Capitalization (44)	Punctuation (44)	Usage (32)	Map Reading (42)	Reading Graphs and Tables (28)	Use of Reference Material (59)	Arithmetic Concepts (48)	Arithmetic Problems (34)	Grand Totals
C 1 -	7th Grade	40	46	25	18	22	19	23	13	34	25	13	278
	8th Grade	36	36	24	14	17	15	25	15	30	19	12	243
C 2 -	7th Grade	26	36	35	26	26	27	15	12	49	29	9	290
	8th Grade	25	47	37	33	28	23	29	11	41	21	16	311
C 3 -	7th Grade	28	43	13	19	18	22	13	15	27	21	12	231
	8th Grade	22	34	17	22	23	16	20	8	19	18	10	209
C 4 -	7th Grade	36	48	41	37	31	25	28	24	51	27	23	371
	8th Grade	33	62	41	37	39	25	31	17	49	28	20	382
C 5 -	7th Grade	34	43	30	25	32	25	19	16	32	18	13	287
	8th Grade	30	40	27	18	22	15	22	17	32	17	12	252
C 6 -	7th Grade	28	37	18	23	18	18	17	20	33	22	14	248
	8th Grade	27	45	21	21	19	16	23	13	36	28	10	259
C 7 -	7th Grade	23	32	21	25	25	22	10	14	32	17	11	232
	8th Grade	23	37	30	28	18	17	16	10	33	18	14	244
C 8 -	7th Grade	27	35	25	14	17	29	10	11	27	14	5	214
	8th Grade	25	40	31	20	20	20	13	11	20	15	6	221
C 9 -	7th Grade	21	32	37	30	29	23	16	19	30	19	12	268
	8th Grade	18	39	37	29	30	18	15	9	42	18	12	267
C10 -	7th Grade	17	29	27	32	20	16	17	12	25	22	9	226
	8th Grade	22	37	23	26	21	16	23	12	42	18	15	255
C11 -	7th Grade	25	35	25	22	24	19	18	16	31	21	12	248
	8th Grade	27	44	23	22	23	20	23	9	40	17	16	264
C12 -	7th Grade	31	49	38	17	31	22	16	11	33	22	11	281
	8th Grade	32	42	34	21	26	16	21	10	40	21	10	273
AVERAGES	7th	28.0	38.8	27.9	24.0	24.4	22.3	16.8	15.3	33.7	21.4	12.0	264.5
	8th	26.7	41.9	28.8	24.3	23.8	18.1	21.8	11.8	35.3	19.8	12.8	265.0

---

*The number of items in the test sections.

TABLE III

SUMMARY OF RAW SCORES BY TEST SECTIONS* — IOWA TESTS OF BASIC SKILLS

Muzzey Junior High School, Lexington, Massachusetts

		7th Grade – October 1968		8th Grade – October 1969		Changes	
	Number of Items	Computer Class	Control Group	Computer Class	Control Group	Computer Class	Control Group
Vocabulary	48	25.0	28.0	25.6	26.7	0.6	-1.3
Reading	80	36.5	38.8	42.3	41.9	5.8	3.1
Spelling	48	25.4	27.9	27.4	28.8	2.0	0.9
Capitalization	44	24.7	24.0	22.3	24.3	-2.4	0.3
Punctuation	44	22.0	24.4	21.0	23.8	-1.0	-0.6
Usage	32	19.9	22.3	16.9	18.1	-3.0	-4.2
Map Reading	42	14.6	16.8	18.6	21.8	4.0	5.0
Reading Graphs and Tables	28	12.4	15.3	12.4	11.8	0	-3.5
Use of Reference Material	59	27.8	33.7	30.5	35.3	2.7	1.6
Arithmetic Concepts	48	21.2	21.4	22.0	19.18	0.8	-1.6
Arithmetic Problems	34	11.8	12.0	11.8	12.8	0	0.8

---

*The raw scores for each section were averaged separately for the 7th and 8th grade tests. The changes above are the differences between these.

Student Performance-Level Changes

At the beginning of the school year, the students in the Lexington junior high schools are placed into one of five tracks, called phases 1 - 5. This is done independently for each of the four subject areas - English, Social Studies, Science, Mathematics. Phase 5 is the most advanced or accelerated group; Phase 1 comprises the children with exceptional learning difficulties. The children in our computer class, and those in the control group, were largely in the middle mathematics track (Phase 3). (Two of the twelve children in the computer class were at the low end of Phase 4. We included them because we thought that the presence of two mathematically more able children might enliven the class. In point of fact, however, two of the Phase 3 children turned out to be the best students, and the Phase 4 children were virtually indistinguishable from the Phase 3 ones in their mathematical work.)

At the end of the year, we recommended a change in placement in mathematics from Phase 3 to Phase 4 for six of the computer class children, and no change for the other six. This was an unusual recommendation: a higher placement for about two out of twelve students is typical. (In fact our judgment was that precisely two out of the twelve children would have been shifted upwards anyway, in a standard mathematics class.)

In the control group, three students advanced from Phase 3 to Phase 4 in mathematics placement. (It should be noted that four of the control group students started the year as Phase 4 math students.) No one in the computer group was down-phased in any subject area; one was in the control group. No one remains in Phase 2 in any subject area in either group. The placements are shown in Table IV.

TABLE IV

PHASE PLACEMENT

Muzzey Junior High School, Lexington, Massachusetts

	Computer Class				Control Group		
Student No.	7th Grade	8th Grade	Change	Student No.	7th Grade	8th Grade	Change
1	3 3 3 4	3 4 4 4	up 2	C 1	4 4 4 4	4 4 4 4	0
2	3 3 3 3	4 4 4 3	up 3	C 2	3 3 4 3	4 3 4 3	up 1
3	3 3 3 3	3 3 3 3	0	C 3	3 3 3 3	3 3 3 4	up 1
4	4 4 3 2	4 4 3 3	up 1	C 4	4 4 4 4	4 4 4 4	0
5	3 3 3 3	3 3 4 3	up 1	C 5	4 4 3 3	4 4 4 4	up 2
6	3 2 3 3	3 3 3 3	up 1	C 6	3 4 4 4	3 3 4 4	down 1
7	4 4 4 4	4 4 4 4	0	C 7	3 3 3 3	4 4 4 4	up 4
8	3 3 3 3	3 3 4 3	up 1	C 8	3 3 3 4	3 3 3 4	0
9	3 3 3 3	4 4 4 3	up 3	C 9	3 3 3 3	3 3 3 3	0
10	3 3 3 3	3 3 3 3	0	C10	3 2 3 3	3 3 4 3	up 2
11	4 4 3 3	4 4 4 3	up 1	C11	3 3 3 4	3 3 3 4	0
12	4 4 4 4	4 4 4 4	0	C12	4 4 3 4	4 4 3 4	0

The series of four digits represents placement in English, Social Studies, Mathematics, and Science in that order. The change (from 7th to 8th grade) is the net sum over the four subjects.

At the beginning of the year, the control group had a somewhat higher placement profile than the computer class, over all subject areas (six more Phase 4 placements, and five fewer Phase 3 placements). At the end of the year, the two profiles were more alike (the control group had three more Phase 4 placements and three fewer Phase 3 placements).

It should again be noted that, since we ourselves made the recommendations for mathematics placement, this is a biased measure.

In early November 1969, we returned to Muzzey Junior High School to talk with the current mathematics teachers of the twelve children who had been in the computer class. We asked how our placement recommendations, all of which had been accepted by the school, were working out. Our students were now divided among three mathematics teachers. Each of the three teachers affirmed that the former computer class students in her class were appropriately placed at present. Two of the new Phase 4 students were among those at the top of their class. They were described as "among the brightest and most enthusiastic, asking lots of questions and volunteering lots of information." The other new Phase 4's were comfortably holding their own. Similarly, the children whose phase placement was unchanged were deemed to be working at the proper level of challenge.

Thus far, our judgment that six students would be capable of doing more difficult mathematics than before (in part as a result of their work with LOGO) appears to be holding up. That is, in personal terms, the most important result of the experiment.

Comments of Evaluators

The following mathematics and science educators independently monitored and evaluated the research throughout the fifteen-month period.

Max Beberman	Professor of Mathematics, University of Illinois Director, University of Illinois Committee on School Mathematics
Robert B. Davis	Professor of Mathematics Syracuse University, and Cornell University Director, The Madison Project
Andrew Gleason	Professor of Mathematics Harvard University Director, Cambridge Conference on School Mathematics
Robert Karplus	Professor of Physics University of California (Berkeley) Director, Science Curriculum Improvement Study

Each of these men made several visits (ranging from three to seven) during the school year to observe the progress of the classroom teaching. They attended two meetings with the project staff just prior to and just after the school year (August 1968 and August 1969), the first one to plan the teaching and evaluation, the last one to discuss the work done during the year.

None of these men were committed to the view of the project staff that programming could make a fundamental contribution to mathematics education. Each man had some familiarity with the ways that computers and programming were already being used in mathematics instruction, and was genuinely interested in our new approach.

Their individual methods of observation and evaluation were different. One of them (Karplus) gave the students his own problems, and on one occasion, his own test. Another (Davis) felt that he had to become personally familiar with the details of LOGO programming and the use of LOGO in teaching. He arranged to get remote access to our computer from his offices at Syracuse and Cornell. Beberman took advantage of his access to local experts on computers and education at Urbana (members of the PLATO instructional project) in planning for his visits to our project. Gleason tested the progress of the class by teaching it himself on occasion. To obtain a further understanding of the problems of designing and using mathematical material in the context of LOGO, he developed some of his own LOGO programs.

Karplus is a physicist; the other three evaluators are mathematicians. We thought it appropriate to have a person whose main concern about mathematics education was its relation to science, and Karplus represented that interest.

At the end of the year, the evaluators prepared statements describing their individual observations and judgments about the project. These are reproduced on the next pages. We begin with Karplus' statement (actually excerpts from several reports that he prepared following each of his four visits during the year). The letter reports of the three mathematicians follow this one.

Letter Reports from Professor Robert Karplus

1. September 18-22, 1968

"Marjorie Bloom promises to be an outstanding teacher for the group. Her knowledge of mathematics and her creative approach to teaching are a rare combination. It is clear that she has many ideas, but that she will be able to use only a few. To keep a record of her work, I recommend that she and one observer write a diary-like report of the procedures and children's reactions after each class. This report should include a brief description of untried alternatives. Even though report writing is burdensome, we have found it to be an essential part of a curriculum study. Human memory is too fallible and too much subject to reinterpretation with the benefit of hindsight.

"Open-ended activities in which a child writes series of instructions for the computer seem to lend themselves quite easily to analysis with regard to the child's level of sophistication. I explained this in connection with the "coded message" activity. I suggest that each child hand in at least one such paper every other week and that you examine it for length, directness and indirectness of the steps in the procedure, etc.

"Some of the operations in LOGO may present logical problems to the children (e.g., class inclusion, transitivity, relationships and their inverses, necessity and sufficiency). Such items should receive special mention in the diary-reports and you should consider modifications in LOGO to make them more clear. You may, for instance, ask the children for the kind of operation they would like to have in LOGO. Their suggestions will indicate their need for redundancy (equivalent basic operations) and/or a different feeling of confidence in basic operations compared to operations which they themselves construct therefrom."

2. December 4-6, 1968

"A great deal of progress was observable during my second class visits on December 4-6, 1968. Six terminals were in operation and the children were skillful in handling them. They dialed to the computer, adjusted the acoustic couplers, and generally conducted themselves in a professional manner. They also used the keyboards with fair accuracy and confidence. In other words, the children's practice in using the terminals is certainly evident in their behavior.

"It was furthermore clear that most of the children can use the LOGO commands to write simple programs, edit them, and use them. Programs with recursion were still a novelty at the time of my visit, but some of the children were catching on to the properties of such a procedure, even though its power was not evident. They enjoyed experimenting with the procedures, finding bugs in programs largely through trial-and-error, and watching the machine operate without their frequent intervention.

"It might be valuable at this point to help the children find some activities that are not ad hoc exercises in LOGO, but actually make use of the program for other purposes. Processing data obtained in science class is one possibility, perhaps generating a forecast of the weekday on which each child's birthday comes for the next twenty years, etc. In other words, the children should begin to think of LOGO as a tool for other purposes, rather than as a self-sufficient and self-contained activity. This aspect can be developed alongside the work on developing programs which are logically more complex."

3. March 27-28, 1969

"I estimate that the children have not moved ahead of their level during my December visit either in skill of teletype operation or in the intellectual level of their program operation. In other words, they are on a plateau as far as their performance is concerned. This plateau is probably related to their general intellectual development and will not be easily surpassed.

"The evidence to support my assertion comes from my observations in the class. The children's principal activities were to de-bug programs which they had typed in accordance with general suggestions or outlines from their teacher. The debugging was still largely a trial-and-error procedure, with very little effort to use a testing strategy for locating the source of trouble. In other words, the binary narrowing-down procedure which was the content of their number-guessing program was not seen as a tool that could be applied in the debugging of programs.

"To help me describe the children's level of reasoning, I asked each one to answer the "Islands Puzzle," a logic quiz. Only one child reasoned that plane connections between A and B, and between A and C, imply a connection between B and C. One of the children based his answers on the islands diagram, seven mostly repeated the information given, and three drew at least partial inferences from the data. This is about par for junior high school students.

"I should like now to go into a couple of background matters. It is useful to distinguish the children's attitude to the program as one of (a) rejection, (b) compliance, (c) identification, or (d) internalization of values. Rejection is self-explanatory; compliance means participation out of compulsion or lack of alternatives; identification means a desire to perform well because of a general appeal of the program, its teacher, or some other connected matter; internalization means an acceptance of the program because of its usefulness to the individual as a tool, for self-expression, etc.

"In my experience, complete internalization of values of any sort is extremely rare in elementary school and probably also in junior high school. Nevertheless, my colleagues and I have the feeling that children should accept the program activities in the same spirit in which we think of them. Perhaps what goes on in your Muzzey computer *club* (though not in your computer class) reflects internalization.

"Identification is the most positive attitude that can be hoped for on any large scale. Whether it is necessary, however, or whether compliant performance results in satisfactory achievement, I don't really know. At present, I would place the Muzzey class between compliance and identification. I can't be more precise, because of variation from child to child and activity to activity. You should really ask yourself how much the attitude factor means to you in comparison with achievement, and whether you are satisfied with the present conditions.

"When students have internalized the teacher's values (graduate school level?), they presumably are eager for advice, leadership, and information from him. When students hover between compliance and identification, this is not the case. A great deal of input from the teacher is likely to result in more compliance or outright rejection; inadequate input from the teacher will lead to wasted time, aimless play, or doing nothing. In other words, the teacher's moves have to be planned and timed carefully, if possible in an individualized fashion. I doubt that this is really possible during the current exploratory phase of your project, but you should make the effort. The children should have enough autonomy so they can test the new ideas in their own ways."

4. May 20-21, 1969

"I was most impressed by the enthusiasm, interest, and obvious competence of the Muzzey computer club members, who raced to the terminals for their turns. Many had designed programs previously (perhaps left from the conference during the prior week) and could hardly wait to try out their ideas.

"The regular computer class was very much more interested and active than it had been during my last visit. Half of the children worked at the terminals (the others were elsewhere in classroom discussion) on various projects. They had been introduced to the concept of generating problems of variable form by using the random number generator in different ways. All the children were quite comfortable in using the random number generator, but some followed the assignment outline quite closely while others were more innovative and still others were occupied with programs unrelated to "variable form". Most of them appeared to be able to control the random numbers adequately. That is, they gave it a particular name when the same random number had to be re-used several times, and they knew that calling for /RANDOM/ would generate a new number."

Summary Report of Professor Max Beberman

"The principal impression I have formed from last year's LOGO experiment is that you are developing a genuinely new approach to mathematics. The ability to construct functions in order to do a given job is a mathematical task of first importance but which is given almost no attention in traditional or new programs. It is important that we see how far into the secondary school we can push this approach. Can we reorganize secondary school mathematics through this approach? Will doing so result in more efficient teaching and learning?

"I gained this impression not from your original proposal [although you may have had this in mind all the time] but through talking with you, watching the children at work, and writing some LOGO programs myself.

"Another characteristic of the experiment which is unique is that children have an immediate feedback on their efforts. They do not need to wait for a teacher to give them the "right answer" in order to check their work. Moreover, there is a real payoff

5

in doing things correctly -- the programs work. In most mathematics classes, students couldn't care less about getting the right answer except as it influences their grades.

"It is difficult to estimate, at this point, what will be the results of the mathematical power acquired by these students. Will it enable them to move through "standard" courses with less difficulty? Will standard courses appear to be trivial?

"As I mentioned on several of my visits, you will have to be creative in finding ways of helping the experimental children attain the standard goals. Rack your brains to find LOGO ways to do this. But it must be done as long as you use public school children in your experiments.

"I am looking forward eagerly to the next steps."

Summary Report of Professor Robert Davis

"May I comment on the Lexington and Newton uses of LOGO by beginning with a discussion of our own uses of LOGO at Cornell University and at Syracuse University. As you know, after we began watching LOGO in use at Lexington, Massachusetts, we were unable to resist some very informal trials of our own. Our trials were in no sense an "experiment," they were just informal exploratory trials intended to give us a feeling for the parameters involved, for possible value, and possible problems.

"By discussing the reasons for our own interest I can provide some background for discussing the work at Newton and Lexington.

"We had eight reasons; four of them relate to what happens in schools and in classrooms:

> 1.) We were looking for ways to get the child's attention. Expecially in some urban classrooms, this is surprisingly hard to do. Both calculators and computers help, and this is not unimportant. Our trials convince us of their value in this regard; I have just conferred with Edith Biggs, one of Her Majesty's Inspectors in England, and Edith reports the same effect there.
>
> If you look at the plight of urban classrooms, you will see what I mean.

2.) We wanted to explore the possibility of allowing children to teach other children. Our trials convinced us that LOGO lends itself readily to such useage. (This, too, can improve the whole climate in a school.)

3.) We try to make some use of a "project" method for letting children learn things. We've tried many things: ideas we got from Edith Biggs, "cardboard carpentry" from E.D.C., etc. Our trials with LOGO make it seem likely that LOGO can provide some worthwhile "projects" for children to develop themselves (what I call "mini-theses").

4.) We've been trying to promote individualization, to make it possible for a teacher to work with one child at a time (or with a small group of 2 or 3 children at a time). LOGO helps, for two reasons: (a) because of its "attention holding" ability, it keeps other children purposefully occupied while the teacher is busy in a different part of the room; and (b) because it has a "quasi - CAI" aspect, it makes it easier for children to work more-or-less on their own. (This rather interesting exploratory "quasi - CAI" arithmetic is very different from straightforward -- and dull -- "programmed-instruction" CAI.)

"One reason relates to our work as a "curriculum-innovation" project:

5.) School procedures and school curricula are frozen so hard that most reasonable efforts fail to yield any change at all. Even worse, there is what Dick Suchman calls the "homeostatic propensity" of curriculum and of school practice -- if you do, miraculously, make some sort of change, before long your resources will usually be exhausted, and both practice and curriculum will return to the status quo ante. We are looking to LOGO to be a powerful enough force for change that it can create change in the first place, and that, in the second place, the new directions will be pursued rather than abandoned.

One could say much about this: it won't automatically happen even from LOGO -- one has to plan for it carefully. Furthermore, if these "new directions" prove to be undesirable, their irreversability suddenly becomes a liability rather than an asset. Many uses of computers do indeed look as if they could become liabilities. LOGO looks to be an asset, and an important one.

"Finally, three reasons relate to the nature of mathematics (or school learning of mathematics):

6.) We want both school children and school teachers to apprehend mathematics as an active thing that you do. It is not merely a lifeless collection of facts and rote procedures. Math is something you think about, you speculate about, you work at. LOGO helps math to appear in this "active" light as "something you do" -- really, as an unlimited collection of things that you can do. Let me emphasize that this view of mathematics is just as important for teachers as it is for children.

7.) We wanted the computer to give us access to problems that are more interesting, but which would be too difficult without a computer. Paul Ward and Dan Aneshansley (at Cornell) have developed some lovely work on population growth. Their simplest model uses essentially only addition; more sophisticated models use Fibonacci series, differential equations, etc. Using a computer puts all of this within reach of average ninth graders, and much of it within reach of 5th and 6th graders (very possibly even younger children could work on it profitably if we got our presentation, introduction and motivation worked out smoothly enough).

Without a computer, these problems would be too tedious to be viable within the school program.

8.) This reason is the most subtle; you would have to observe 7th, 8th, and 9th grade classrooms to see what it really means. I have watched a teacher devote 40 minutes to problems, all of the same type, as follows:

"How shall we write the sum of twice A and B?"

(Ans: 2A + B)

This was a wildly dull lesson; an unnecessary one; and an unproductive one. This teacher complained because she never had enough time to teach all the math that people expected her to teach.

A second example: Asked to "solve a + a = 10 for a," a child responded "For which a?"

Now these examples are a vague suggestion of what seems to be an important problem. Obviously, the problem has complex roots. Some of its roots lie in the theory that the teacher

> is supposed to tell the student what to do, and how to do it. But much of the difficulty probably lies here: the thing being discussed is the written language of mathematics as it is usually handled in 9th grade algebra. In order to discuss this, a certain ill-defined and wildly inelegant language is being used. In fact, most of the content of traditional algebra courses consists precisely in learning this language. It is a poor language, and a waste of time to learn. We no longer teach the historically earlier forms of this language ("If the ten-fold multiple of a quantity be diminished by one half of its cube, the result is to 24 as the square of 3 is to its third power"). We need to stop teaching the 1969 version, which doesn't justify itself.
>
> This requires something else to be put in its place. The Carbondale group (Exner, et al.) are experimenting with formal mathematical logic. You are experimenting with LOGO. (We have been experimenting with the use of as little language as possible, and that little made up mostly by the children themselves.) Obviously, I hope we all win. But we, too, see this use of LOGO as potentially valuable.

"So much for our informal explorations with LOGO. (We have worked with teachers of grades K - 9, with prospective teachers, and with children from age 3 years to age 21 years, or so.)

"I should add that I was present at the Joint Computer Conference in Boston last year when Sharon Kaufman, M.D., a psychiatrist from U.C.L.A., observed your Newton and Lexington children at work and described the activity from her (very valuable) point of view: in terms of children learning from errors instead of being discouraged by them, instead of being driven into uncontrollable frustration by them, etc. She spoke in terms of frustration tolerance, perseverance, flexibility, etc. -- and expressed the opinion that these traits, perhaps more than any others, determine whether one leads a fruitful life or becomes a social reject. I am impressed with her remarks, they seem consistent with our experience, but I can describe things better from my usual frame of reference in relation to school mathematics.

"Let me now comment on the work at Lexington and Newton.

> i) Obviously, I like it -- in fact, I was deeply impressed by it -- or I and my colleagues at Cornell would not have gone to great lengths to explore it ourselves.

ii) I am impressed by such facts as these: the children showed normal gains on the Iowa Tests even though this is not what they studied in grade seven: LOGO was their (essentially entire) math program for seventh grade, yet they showed normal growth on the (essentially obsolete) Iowa tests. Also, you had 12 essentially randomly-selected middle track (or, as they say in Lexington, "phase") students. For grade 8, none has dropped below his expected track, but 6 students -- one-half of the class -- have moved up to a "higher," "better," or "more difficult" track. This is very important, because a major concern in education today is the fact that once you consign a child to a lower track, you have put him there for the rest of his life. Your results in this regard are both very unusual and potentially very important.

"Let me remark upon Lexington as an "experiment" in relation to the conventional wisdom of educational research. We both know that it was not an "experiment" in the conventional sense. You received excellent advice, and deliberately chose a different route. I think that was wise.

"But what (if any) systematic or generalizable knowledge could be gained from future similar explorations? Some of my friends say: none -- learning ecology doesn't lend itself to generalizable data any better than painting or music. They may be right. Nonetheless, several things seemed clear:

i) I observed 3 different teachers working with the 7th graders. Teachers style, personality, approach to mathematics and approach to the children were so different that they weren't even in the same ball-park. You didn't have one experiment, you had three (not to mention interaction effects among the three different treatments).

ii) While you had less variation in classroom format, time scheduling, etc., the same potential for variation was there.

"It is almost as if there were no such thing as "teaching LOGO to seventh graders," no more than there is such a thing as "playing the piano." My six-year-old daughter practicing, one hand at a time, is one thing; her five-year-old brother banging with both hands is something else (and more musically rewarding for the listener). Glen Gould's Bach is different from Wanda Landowska's (she played piano, too, sometimes!) Bach is different from Chopin, and so on. There is so much potential (and actual) variability that we need a new approach. I think it not inconceivable that art criticism may offer us better paradigms than seventeenth century science does, or than studies of the effectiveness of different fertilizers.

"As in nearly everything human, how you do it is as important as what you do. The message is in fact inseparable from the medium.

"I can't imagine that anybody doubts that LOGO is a good thing. Bach is a good thing. Beethoven is a good thing. The Cauchy - Riemann equations are a good thing. LOGO is a good thing.

"And, unfortunately, we can't cop out and say -- well, it's all a matter of cost effectiveness. Building Edsels is a matter of cost effectiveness; education isn't."

Summary Report of Professor Andrew M. Gleason

"My impressions of the LOGO project: I visited the classes on three occasions last year.

"The first time I was not particularly impressed. It seemed that progress was slow and the children didn't show much in the way of disciplined interest.

"By the second visit interest had improved considerably. On that occasion I led a discussion of possible algorithms for multiplication. The children had done a little with this beforehand and they were clearly involved in finding ways to do it better. I understand that subsequently they did a good bit of work improving their algorithms and doing division.

"The last time I came there were some really impressive things going on. The children had learned to make a random sentence generator and some were trying to improve theirs. Aside from the higher level of programming sophistication involved, there is a clear opening into an entirely different area of learning. To program the machine to make up English sentences requires that they look critically at what distinguishes an English sentence from a string of words. This is what grammar is all about. Approaching grammar through a computer context, the children are much more likely to see it as relevant to the important problem of communication, than they are in an English class where the emphasis is more likely to be on whether a sentence is "good" English than whether it carries the message it intends.

"The same day there was a girl who was programming a French verb conjugator. Another example of transfer to other areas. Who would have thought that your project would be reinforcing some girl's knowledge of French verbs?

"It seems to me that these carryovers are the really important contribution of the program. One can hardly expect that each child should be interested in school work in all areas. We shall be very successful indeed in our schools if we can get each child to think seriously about one topic. The computer, as expected, does seem to challenge students to make a real intellectual effort. At the same time its versatility offers a much larger range of attractive problems than is available under conventional programs. These problems are self-adapting; a child can choose his own problem and push it as far as he pleases, there being almost no limit to the scope of computer applicability.

"I think computers in the classroom will continue to open new vistas in education and increase the probability that children will learn the one fundamental lesson, that serious thought can pay off in increased insight into almost any problem."

Conclusions of Project Staff

Mathematical Value

The main result of the research and teaching has been to uncover an abundance of ideas for teaching, only a fraction of which had been anticipated at the time we started the project. The experiment showed also that LOGO can be used to express a wide diversity in teaching styles and modes of presentation.

Side Effects

There are indications that the childrens' work with LOGO directly contributed other educational and behavioral benefits. There was evident enhancement in reading rates of some second and third grade children. Administrators and teachers in the junior high school stated that behavioral changes in certain of the children (such as a higher degree of self-confidence, more positive social attitudes, broadening of intellectual interests, etc.) were attributable to their experience in the course.

Feasibility

We are not sure about these extra-mathematical benefits. But, we are sure of the following two points:

> It is feasible to teach LOGO to "average" seventh-grade, and younger, children.
>
> It is feasible, also, to develop and effectively teach a mathematics curriculum using LOGO as the conceptual and operational framework.

Problems

The day-to-day work during last year consisted in testing very specific ideas intended to achieve these goals. As part of this work, we identified conceptual gaps in the children, some of which were surprising in their obviousness once we had recognized them. Typical examples of such missing concepts are:

the absence of any idea like "syntactic form," despite the time spent on formal manipulation in the mathematics classroom and on analyzing the structure of sentences in language classrooms;

the rareness of the idea of "counter-example" and the complete absence of any systematic habits of looking for counter-examples to test ideas;

an *inability* to go into a "formal mode" of thinking, i.e., *following quite literally* a set of instructions or definitions;

the absence of an idea of a planning phase of work on a problem.

Once these issues are identified, it is *relatively* easy to develop appropriate teaching materials, modes of presentation, well-sequenced units, and projects. This is where further work needs to go.

APPENDIX: A Description of the LOGO Language and System

1. The LOGO Language

1.1 Things, Operations, Commands, and Names

There are two kinds of LOGO things -- WORDS and SENTENCES.

WORDS

Examples of LOGO WORDS:

"SUN"	(an English word)
"39"	(a numerical word)
"PFFS!T!220W*?"	(a nonsense word)
""	(a special word called the EMPTY word)

SENTENCES

A LOGO SENTENCE is a series of LOGO words, separated by spaces. (LOGO words do not contain spaces.)
Examples:
"GOOD MORNING"
"X + Y = 24.5"
"FLOOCH HUM BUZZ CHORB"

OPERATIONS

LOGO has several elementary (i.e., built-in) OPERATIONS for manipulating LOGO things. A LOGO operation takes a fixed number of things (possibly none) as INPUTS and produces a new thing as an OUTPUT. Examples of some built-in operations-FIRST, LAST, BUTFIRST, BUTLAST, WORD and SENTENCE-follow.

FIRST

The operation FIRST takes one input, either a word or a sentence. Its output is the first letter (if the input is a word), or the first word (if the input is a sentence). Thus,

FIRST OF "CAT" is "C"

and

FIRST OF "DO RE MI" is "DO".

LAST, BUTFIRST, and BUTLAST

These operations are similar to FIRST, as the following examples show.

Name of Operation	Input	Output
LAST	"CAT"	"T"
BUTFIRST	"CAT"	"AT"
BUTLAST	"CAT"	"CA"
LAST	"DO RE MI"	"MI"
BUTFIRST	"DO RE MI"	"RE MI"
BUTLAST	"DO RE MI"	"DO RE"

WORD and SENTENCE

These operations take two inputs. Their output is the concatenation of their inputs. The inputs of WORD must be LOGO words, not sentences. The output of WORD is a LOGO word. Thus, WORD OF "DO" AND "RE" is "DORE". The inputs of SENTENCE can be LOGO words or LOGO sentences. The output is a LOGO sentence. Thus,

Name of Operation	Inputs	Output
WORD	"TIC" "TAC"	"TICTAC"
SENTENCE	"TIC" "TAC"	"TIC TAC"
SENTENCE	"PUT ME" "HERE"	"PUT ME HERE"
WORD	"PUT ME" "HERE"	(Error Message)

The entire set of elementary, i.e., built-in, operations is described in Section 1.7.

CHAINING

Operations can be chained together to form composite operations. Examples:

Chained Operation	Output
FIRST OF BUTFIRST OF "CAT"	"A"
LAST OF FIRST OF "DO RE MI"	"O"
WORD OF BUTLAST OF "CAT" AND LAST OF "X9"	"CA9"
SENTENCE OF WORD OF "A" AND "B" AND "C"	"AB C"
WORD OF "Z" AND SUM OF "1" AND "2"	"Z3"

COMMANDS

LOGO has several built-in COMMANDS. Commands have inputs but, unlike operations, do not make a new LOGO thing, i.e., they have no output. They are used for their external effects or for their side effects.

PRINT is a built-in LOGO command which has one input. Though it has no output, it has the tangible effect of causing its input to be printed out by the teletype. Thus,

Command	Input	Printout
PRINT	"CAT"	CAT
PRINT	LAST OF "BOX"	X

Other built-in LOGO commands include MAKE which is used to give names to LOGO things, and TO which is used to define new LOGO operations and commands. These are described in later paragraphs of this section.

LITERALS

In the examples above, some words and sentences are enclosed in quotation marks. This means that we are citing them literally, to refer to themselves rather than to other LOGO things. But we also can use LOGO things as names for other LOGO things.

NAMES

LOGO things can have NAMES. Any LOGO thing (except the empty word) can be used as a name for any other LOGO thing. Assume that we have assigned "JANE" as the name of the thing "GIRL". We now can use "JANE" in two ways - either as a thing (itself) or as a name (for the thing "GIRL").

To indicate to LOGO that we want to use something as a name (in order to refer to the thing that it names), we have a LOGO operation - THING, whose input is a LOGO thing and whose output is the LOGO thing named by the input. Thus, with the example above

Command	Printout
PRINT "JANE"	JANE
PRINT THING OF "JANE"	GIRL

A shorthand way of writing THING OF "ANYTHING" is /ANYTHING/. Thus, the effect of the command PRINT /JANE/ is to cause the teletype to print GIRL.

NAMING

The LOGO command MAKE is used to construct a LOGO THING and give it a NAME. The following example shows how a student could use it to assign "JANE" as the name of "GIRL".

```
MAKE
   NAME:  "JANE"
   THING:  "GIRL"
```

The student's typing is underscored to distinguish it from LOGO's responses.

More typically, one names a more complex construction, as follows:

```
MAKE
   NAME:  "JIM"
   THING:  LAST OF BUTLAST OF /JANE/
```

In this instance, with /JANE/ assigned as above to "GIRL", /JIM/ would name the THING "R" (since BUTLAST of /JANE/ is "GIR" and LAST of "GIR" is "R").

Constructing a New Operation

Suppose we want to define an operation which has one input and whose output is the second letter of its input (if the input is a word) or the second word (if the input is a sentence). We will call this new operation SECOND. (We can choose any word not already being used by LOGO as a procedure name.) The procedure for performing SECOND is described to LOGO as follows.

```
TO SECOND /ANYTHING/
1  OUTPUT FIRST OF BUTFIRST OF /ANYTHING/
END
```

TO is a command that signals the start of a procedure definition. The name of the procedure we are defining is SECOND. Its input is /ANYTHING/, which names the LOGO thing we will operate upon. It has a single instruction line (in general there are several), labeled 1. The instruction is: OUTPUT the thing expressed by the chained operation FIRST OF BUTFIRST OF the word (or sentence) named by /ANYTHING/. END demarcates the end of the procedure definition.

SECOND is now the procedure name for a procedure which defines a new operation. To perform this new, user-defined (as distinct from elementary or built-in) operation, we can give LOGO the command

```
PRINT SECOND OF "MAN".
```

This tells LOGO to perform the operation SECOND on the input "MAN", i.e., that /ANYTHING/ is now "MAN". LOGO will perform the instructions in the procedure. It will thus output "A" to the PRINT command which will cause the teletype to print A.

1.2 Instructions

The basic unit of a LOGO INSTRUCTION is a LOGO EXPRESSION. An expression has two parts: (1) a procedure name or the name of an elementary (built-in) command or operation, followed by (2) a list of the associated inputs. Some examples of expressions are:

(a) WORD OF "CAT" AND "DOG"
WORD is a built-in LOGO operation that requires two inputs, in this case "CAT" and "DOG". The output of this expression is the word "CATDOG".

(b) SECOND OF "APPLE PIE SOUFFLE"
SECOND is a procedure defined by the user which requires one input, in this case, "APPLE PIE SOUFFLE". Assuming that the procedure is defined as in the previous section, the output of the expression would be "PIE".

(c) TIME
TIME is a built-in operation that requires no inputs. The output of this expression is the current time, for example, "10:34 AM".

The inputs in expressions may be LOGO NAMES as well as LITERALS.

(d) FIRST OF /CHILDREN/
FIRST is a built-in operation that requires one input. The input here is not "CHILDREN" but rather the thing that "CHILDREN" names. Thus, if "CHILDREN" is the name for the LOGO sentence "BOYS AND GIRLS", the output of the expression is "BOYS".

The inputs in expressions may themselves be expressions.

(e) FIRST OF BUTFIRST OF "ABCD"
Here the input of the operation FIRST is the output of the expression BUTFIRST OF "ABCD", that is "BCD". So the output of the whole expression is the same as the output of FIRST OF "BCD", that is "B".

Commands are expressions. For example,

(f) PRINT OF "ABC"
This expression has no output but it causes ABC to be printed by the teletype.

(g) PRINT OF FIRST OF "ABC"
In this expression, PRINT has, as its input, the output of FIRST OF "ABC", that is "A". The effect is to cause A to be printed by the teletype.

On the other hand, the form
(h) FIRST OF PRINT OF "ABC"
is not a legal expression because FIRST requires an input but the expression PRINT OF "ABC" has no output.

In writing expressions, the words OF and AND are optional. The expressions PRINT WORD "CAT" "DOG", PRINT OF WORD OF "CAT" AND "DOG", and PRINT WORD OF "CAT" AND "DOG" all have the same meaning.

1.3 Procedures

Several LOGO instructions can be put together to form a PROCEDURE. This is accomplished using the instruction TO. (The student's typing is underscored to distinguish it from the computer's.)

```
←TO GREET /NAME/
>10 PRINT SENTENCE OF "HELLO," AND /NAME/
>20 PRINT "I HOPE YOU'RE WELL."
>END
GREET DEFINED

←GREET "DICK"
HELLO, DICK
I HOPE YOU'RE WELL.
```

The instruction in first line of the example, TO GREET /NAME/ does several things. The word TO tells the computer that we are about to define a procedure. The next word GREET is the name of the procedure. Following this is the list of input names for the procedure, in this case only one. (If we had wanted a two-input procedure, like WORD, the first line might have been TO GREET /WHO/ AND /WHERE/. Any number of inputs is permitted including none.) The names appearing in the input list are used in subsequent instructions to refer to the associated inputs.

After the TO instruction (also called the title line of the procedure), the computer types > at the beginning of each line to indicate that it is ready for the type-in of the next line of the procedure being defined. At this point any line typed in preceded by a number (between 1 and 999999 inclusive), as lines 10 and 20 in GREET, will be made part of the procedure definition. These instructions will subsequently be performed in the numerical sequence thus indicated. The instructions are not performed immediately as they are written - they are merely stored as part of the definition. They can be performed later when the procedure definition has been completed.

Finally, the command END (which has no inputs) completes the definition of the procedure. The computer types GREET DEFINED. Now the computer begins lines with an ← indicating that it is ready to perform an instruction (possibly a procedure).

GREET may now be used as a LOGO command. The expression GREET "DICK" (or GREET OF "DICK") causes the computer to pair the input, "DICK" with the name in the title line of GREET, /NAME/, and then to carry out the instructions in the body of the procedure GREET in the numerical order of their line numbers.

There are two ways to change the numerical order of execution of the instructions in a procedure. The first is by the command

GO TO LINE expression

where the value of expression must be a line number. (Although the name of this command is a sentence, GO TO LINE, it is a single entity of the same kind as commands whose names are single words.) The effect of this command is shown in the following example.

```
←TO SHOWGOTOLINE /X/
>10 GO TO LINE /X/
>20 PRINT "1"
>30 PRINT "2"
>40 PRINT "3"
>50 PRINT "4"
>END
SHOWGOTOLINE DEFINED
>SHOWGOTOLINE "50"
4
>SHOWGOTOLINE "30"
2
3
4
```

```
←SHOWGOTOLINE "25"

THERE IS NO LINE 25                     (LOGO types out these
I WAS AT LINE 10 IN SHOWGOTOLINE        diagnostic comments.)

←SHOWGOTOLINE "10"
    (interrupt key pressed here after some time has gone
    by with no printout)
I WAS AT LINE 10 IN SHOWGOTOLINE
```

In the above example SHOWGOTOLINE "10" caused the computer to do line 10 over and over again until it was interrupted from the teletype. A more standard example of the use of GO TO LINE is as follows.

```
←TO TWOTIMES
>10 MAKE
    NAME:  "X"
    THING:  "1"
>20 PRINT /X/
>30 MAKE
    NAME:  "X"
    THING:  SUM OF /X/ AND /X/
>40 GO TO LINE 20
>END
TWOTIMES DEFINED

←TWO TIMES
1
2
4
8
16
32
64
  (interrupted from teletype)
I WAS AT LINE 20 IN TWOTIMES
```

The other way of altering the numerical order of execution of the instructions in a procedure is with the trio of commands TEST, IF TRUE, and IF FALSE. TEST takes one input, which must be an operation whose output must be either "TRUE" or "FALSE". (TEST can also take as input the literal words "TRUE" and "FALSE".) The effect of performing the command TEST <u>expression</u> is to mark a "truth flag" either true or false, depending on whether the output of <u>expression</u> is "TRUE" or "FALSE", respectively.

IF TRUE and IF FALSE are somewhat anomalous commands in that their input can be any instruction, even a command. That instruction is executed if the truth flag matches the second word of the IF --- command.

```
←TEST "TRUE"
>IF TRUE PRINT "OF COURSE"
OF COURSE
←IF FALSE PRINT "STRANGE"
     (No printout occurs since the truth flag is
      marked TRUE)

←TO SHOWTEST /X/
>10 TEST /X/
>20 IF TRUE PRINT "AXLE"
>30 IF FALSE PRINT "CAKE"
>40 IF TRUE PRINT "SUBWAY"
>END
SHOWTEST DEFINED
←SHOWTEST LAST OF "BLUE TRUE"
AXLE
SUBWAY
←SHOWTEST FIRST OF "FALSE LOVE"
CAKE
←
```

The IF commands can be used with GO TO LINE instructions to provide conditional branching within a procedure. More broadly, they can be used to alter the sequence of execution of procedures within a program comprising many procedures (as in 30 IF TRUE TARUM - 40 IF FALSE TARAY where the condition of the truth flag determines whether the computer does the procedure TARUM or the procedure TARAY).

TEST is made more useful by a collection of built-in operations which output either "TRUE" or "FALSE". Operations which can have only these two values are called predicates. Section 1.7 includes a list of the built-in predicates.

1.4 Defined Operations

It is possible to define new LOGO operations (i.e., procedures which have an output) by means of the command OUTPUT.

```
←TO DOUBLE /X/
>10 OUTPUT WORD OF /X/ AND /X/
>END
DOUBLE DEFINED
←PRINT DOUBLE OF "CAT"
CATCAT
←PRINT DOUBLE OF DOUBLE OF "GO"
GOGOGOGO
←
```

An apparently equivalent procedure that doesn't have an output is

```
←TO DUB /X/
>10 PRINT WORD OF /X/ AND /X/
>END
DUB DEFINED
←DUB "CAT"
CATCAT
←
```

Notice that it wasn't necessary to say PRINT DUB OF "CAT" since DUB contains a PRINT command. (The appearance and disappearance of the OF is purely for euphony. The computer ignores it.) What if an external PRINT is used?

```
←PRINT DUB OF "CAT"
CATCAT

DUB CAN'T BE USED AS AN INPUT.  IT DOESN'T HAVE AN OUTPUT.
←
```

LOGO complains. The problem is that the external PRINT didn't get any input because DUB didn't output anything -- DUB is a command, not an operation. The word "CATCAT" got printed anyway because that happens before the computer gets to the end of DUB and discovers that there is no output to transmit to the external PRINT command.

The same thing happens, giving an obviously wrong answer, when one writes

```
←DUB DUB "GO"
GOGO

DUB CAN'T BE USED AS AN INPUT.  IT DOESN'T HAVE AN OUTPUT.
←
```

Once procedures like DOUBLE or DUB are defined, they are virtually indistinguishable in their use from the built-in operations and commands. Thus, in the same way as with the built-in ones, these too can be used to define other procedures.

```
←TO TRIPLE /X/
>10 OUTPUT WORD OF /X/ AND DOUBLE OF /X/
>END
TRIPLE DEFINED
←PRINT TRIPLE OF "AB"
ABABAB
←PRINT TRIPLE OF DOUBLE OF "R"
RRRRRR
←
```

1.5 Recursion

In fact, since a defined procedure can be used just like a built-in procedure, it can even be used in its own definition. Sometimes this gets nowhere -

```
←TO TYPEALOT
>10 TYPEALOT
>END
TYPEALOT DEFINED
>TYPEALOT
     (after a long wait the interrupt key is hit)
I WAS AT LINE 10 IN TYPEALOT
←
```

It was silly to expect the computer to have been able to perform this procedure (to type a lot?) with the instructions we gave. If it didn't know what TYPEALOT meant before we defined it, it certainly wouldn't now. But it clearly was doing something when we said TYPEALOT since the teletype didn't type an ← or an error message.

When the computer receives the instruction TYPEALOT, it sees that the instruction names a defined procedure. In order to perform it, it has to look up the instructions contained in the procedure definition. The title line shows that no inputs are needed. Then the next line tells the computer to perform the procedure TYPEALOT. To do this, the computer must look up the procedure TYPEALOT and then perform the instructions contained there. When it does this, it once more finds that it must look up the procedure TYPEALOT, all over again. And again and again. And so this goes on forever. (Actually, the LOGO system will assume that there is an error after it has looked up this procedure about 500 times, and will then cause the computer to stop.)

Of course, it would have been easy to design LOGO so that it would remember what procedure it was doing and not allow this situation to occur. It turns out, though, that we would have deprived ourselves of a very valuable mathematical tool had we done this.

The simplest use of the above effect (called recursion because of the recurrence of the same definition) is to note that if there had been a line preceding line 10 in the procedure TYPEALOT, this line would be done over and over again, every time the procedure is looked up. Let us add a new line, say line 5.

```
←TO TYPEALOT
>5 PRINT "A LITTLE"
>10 TYPEALOT
>END
TYPEALOT DEFINED
←

←TYPEALOT
A LITTLE
A LITTLE
A LITTLE
    .
    .
    .
        (the interrupt key is hit to stop it)
I WAS AT LINE 5 IN TYPEALOT.
←
```

The following recursive procedure takes an input and has a stopping rule.

```
←TO TRIANGLE /ANYWORD/
>10 TEST EMPTYP OF /ANYWORD/
>20 IF TRUE STOP
>30 PRINT /ANYWORD/
>40 TRIANGLE BUTFIRST OF /ANYWORD/
>END
TRIANGLE DEFINED
←
```

EMPTYP is a built-in predicate operation that outputs "TRUE" if its input is the empty word and outputs "FALSE" otherwise. STOP is a built in command to stop this procedure, i.e., to skip the rest of the instructions in the procedure and go directly to the end.

```
←TRIANGLE ""    (trying TRIANGLE with the empty word as the input)
←               (nothing printed out but the program stopped)
←TRIANGLE "ABCDE"
ABCDE
BCDE
CDE
DE
E
←
```

In the second example, the definition of TRIANGLE was looked up six times. The first five times the input was not the empty word, so the STOP command was skipped. The computer then typed the input and looked up TRIANGLE again, but this time with a smaller input (the butfirst of the previous one). Finally, the input was the empty word. For that input TRIANGLE skips lines 30 and 40 (so nothing is typed and the procedure is not looked up again) and it stops.

A well known example of this type of definition in arithmetic is the one for factorial:

$n! = 1$ if $n=1$, otherwise $n! = n \cdot (n-1)!$

This can be transcribed directly to LOGO.

```
←TO FACTORIAL /N/
>10 TEST IS /N/ "1"
>20 IF TRUE OUTPUT "1"
>30 IF FALSE OUTPUT PRODUCT OF /N/ AND FACTORIAL OF
          DIFFERENCE OF /N/ AND "1"
>END
FACTORIAL DEFINED
←
```

IS is a built-in predicate that outputs "TRUE" if its two inputs are expressions for the same thing and "FALSE" otherwise. Line 30 is rather long but not too hard to read if one uses parentheses PRODUCT OF (/N/) AND (FACTORIAL OF [DIFFERENCE OF /N/ AND "1"]).

DIFFERENCE OF /N/ AND "1" is just n-1.

PRODUCT isn't a built-in operation so this FACTORIAL procedure will not actually work until we also write a procedure PRODUCT. Finally, it is not necessary to prefix the instruction in line 30 with the command IF FALSE, since the OUTPUT command incorporates the actions of the STOP command and, if line 20 is executed, the OUTPUT command there will skip to the end of the procedure.

Here is a PRODUCT procedure based on Peano's definition of multiplication.

```
←TO PRODUCT /X/ AND /Y/
>10 TEST IS /X/ "1"
>20 IF TRUE OUTPUT /Y/
>30 IF FALSE OUTPUT SUM OF (/Y/) AND (PRODUCT OF
        [DIFFERENCE OF /X/ AND "1"] AND /Y/)
>END
PRODUCT DEFINED
←PRINT PRODUCT "3" AND "12"
36
←PRINT FACTORIAL "5"
120
```

1.6 Local and Global Names

In LOGO everything except the empty word is the name of something. Until they are otherwise assigned, almost all LOGO things name the empty word.

```
←PRINT /SOMETHING/
    (The computer prints the empty word by skipping a line.)
←PRINT /ANY OTHER THING/

←PRINT /A/
```

```
←PRINT //

THE EMPTY WORD CANNOT BE A NAME.

←
```

The few exceptions, which don't initially name the empty word, are built-in LOGO names for special things such as the teletype bell, the blank character, etc. These are listed in Part 3.

Names may have their things changed in two ways. The most direct way is by means of the instruction MAKE.

```
←MAKE
   NAME:  "ALPHA"
   THING:  "BETTY"
←PRINT /ALPHA/
BETTY
```

The text following the words NAME: and THING: may be anything that has an output, that is a literal (like "ALPHA"), a name (like /JKS/), or an operation with its inputs.

```
←MAKE
   NAME:  /ALPHA/
   THING:  "SAPLE"
←PRINT /BETTY/
SAPLE
←
```

The name here is /ALPHA/, that is the LOGO thing "BETTY".

```
←MAKE
   NAME:  SENTENCE OF "DOT" AND /ALPHA/
   THING:  BUTFIRST OF /BETTY/
←PRINT /DOT BETTY/
APLE
```

Here the name of SENTENCE OF "DOT" AND /ALPHA/ which is "DOT BETTY" and the thing it names is BUTFIRST OF /BETTY/, "APLE".

The instruction LIST ALL NAMES causes all names with non-empty things to be listed.

```
←LIST ALL NAMES
/ALPHA/ IS "BETTY"
/BETTY/ IS "SAPLE"
/DOT BETTY/ IS "APLE"
←
```

Just as the OF and AND in most instructions are optional, the carriage returns after the command MAKE and before the label THING are optional. The instruction in the form

```
←MAKE  (carriage return)
   NAME:  "BB"  (carriage return)
   THING:  "CABF"  (carriage return)
```

can also be written with the two inputs on one line. Thus:

```
←MAKE "BB" "CABF"  (carriage return)
```

The computer doesn't type out NAME: and THING: in this form so it is a little faster to type in.

The slow form is useful in emphasizing the relation between NAME and THING during the early stages of teaching, however.

The other method of changing names is by specifying inputs in procedures.

```
←TO SHOW /X/
>10 PRINT "NOW I AM GOING TO PRINT /X/"
>20 PRINT /X/
>END
>SHOW "CATS"
NOW I AM GOING TO PRINT /X/
CATS
←
```

While the procedure SHOW is running, /X/ stands for "CATS". When it stops, however, the old THING OF "X" is restored. Thus,

```
←PRINT /X/
          (the empty word)
←MAKE
   NAME:  "X"
   THING:  "OLD THING"
←PRINT /X/
OLD THING
←SHOW "DOGS"
NOW I AM GOING TO PRINT /X/
DOGS
←PRINT /X/
OLD THING
←
```

A name which is in force only during the running time of some procedure is called local to that procedure. A name that isn't local to any procedure is called global. In the example above, /X/ ("OLD THING") was global, while /X/ ("DOGS") was local to SHOW. While a local /X/ is in force (i.e., while the procedure for which it is local is running), all references to /X/ as a name refer to the local name.

```
←TO WORRY /X/
>10 PRINT /X/
>30 PRINT THING OF "X"
>40 MAKE
      NAME:  "X"
      THING:  WORD OF "CAT" AND /X/
>50 PRINT /X/
>END
WORRY DEFINED
←PRINT /X/
OLD THING
←WORRY "PIPE"
PIPE
PIPE
CATPIPE
←PRINT /X/
OLD THING
←
```

One reason for this somewhat complicated situation is that it permits the student to forget about the choice of names inside of procedures that he has written. For example, suppose the student had written a procedure to output the product of two numbers and it had a title line TO PRODUCT /X/ AND /Y/. Then, sometime later he wrote another program, called TO QUADRATIC /X/, for computing (X+1)X+3X, which uses the procedure PRODUCT in its definition.

```
←TO QUADRATIC /X/
>10 MAKE
      NAME:  "FIRST TERM"
      THING:  PRODUCT OF (SUM OF /X/ AND "1") AND /X/
>20 MAKE
      NAME:  "SECOND TERM"
      THING:  PRODUCT OF "3" AND /X/
>30 OUTPUT SUM OF /FIRST TERM/ AND /SECOND TERM/
>END
QUADRATIC DEFINED
←PRINT QUADRATIC OF "4"
32
←
```

Notice that /X/ becomes "4" on entering QUADRATIC. In line 10 however PRODUCT OF "5" AND "4" is evaluated and so PRODUCT is run. But that causes /X/ to become "5" while PRODUCT is running. When PRODUCT is finished, however, we come back to QUADRATIC, now at line 20 and see another reference to /X/, meaning the /X/ of QUADRATIC ("4"). And, indeed, this is the way things work because the /X/ in PRODUCT is local to that procedure and disappears when PRODUCT is finished, leaving the /X/ of QUADRATIC in force.

In this case the problem could have been gotten around simply by using different input names for all procedures, a possible, if awkward, maneuver. There is an important case where that won't work, though, and that is in recursive procedures. There, since the procedure being called is the same procedure as the one being run, the input names are, of course, identical. Here is an example of a recursive procedure that doesn't work properly because some global names are treated as though they were local.

```
←TO REVERSE /X/
>10 TEST EMPTYP OF /X/
>20 IF TRUE OUTPUT /EMPTY/
>30 MAKE
        NAME:  "NEW BEGINNING"
        THING:  LAST OF /X/
>40 MAKE
        NAME:  "NEW END"
        THING:  REVERSE OF BUTLAST OF /X/
>50 OUTPUT WORD OF /NEW BEGINNING/ AND /NEW END/
>END
REVERSE DEFINED
←PRINT REVERSE OF "CAT"
TTT
←
```

The problem here is at line 30 and at line 40. /NEW BEGINNING/ isn't an input to REVERSE, so it isn't local. Therefore, its thing will change on subsequent calls of REVERSE. The computer does not save the things of global names in each round - it only does that for local names. (The same is true for /NEW END/ though in this procedure that doesn't affect the result -- nothing that might change /NEW END/, such as a call to REVERSE, happens after line 40 where /NEW END/ is set.) At line 40 another REVERSE is called. In executing this REVERSE procedure, /NEW BEGINNING/ will change. REVERSE procedures can, of course, be written to avoid this problem. But this REVERSE procedure can easily be repaired by making /NEW BEGINNING/ local to it. There is a command, LOCAL, to do this. LOCAL takes one input, the name that is to be made local to the procedure.

```
←TO REVERSE /X/
>5 LOCAL "NEW BEGINNING"
>10 TEST EMPTYP OF /X/
         :    (same as before)
         :
>END
REVERSE DEFINED
←PRINT REVERSE OF "CAT"
TAC
←
```

1.7 List of Elementary Operations

1. FIRST (one input)

Its output is the first word of a sentence or the first letter of a word.

FIRST OF "AB12X!" is "A"

FIRST OF "MOX SED PEAX" is "MOX"

2. LAST (one input)

Its output is the last word of a sentence or the last letter of a word; analogous to FIRST.

3. BUTFIRST (one input)

Its output is all but the first word of a sentence or all but the first letter of a word.

BUTFIRST OF "AB12X!" is "B12X!"

BUTFIRST OF "MOX SED PEAX" is "SED PEAX"

There is one tricky point here. BUTFIRST of a two-word sentence is the last word of the sentence. It is a one-word sentence, however, not a word. This can be observed in the expression FIRST OF BUTFIRST OF "THE DOG" which has as its output the word "DOG" since that is the first word of the one-word sentence "DOG" that is the output of BUTFIRST OF "THE DOG". Continuing further, the output of FIRST OF FIRST OF BUTFIRST OF "THE DOG" is the word "D". In practice the output type (word or sentence) almost always works out as the user expects.

4. BUTLAST (one input)

Analogous to BUTFIRST.

(It is worth noting that the output of BUTFIRST or BUTLAST is the same type (word or sentence) as its input. On the other hand, the output of FIRST or LAST is always a word.)

5. WORD (two inputs)

Both inputs must be words. The output of the expression is a new word made by concatenating the two inputs.

WORD OF "MO" AND "ZART" is "MOZART".

6. SENTENCE (two inputs)

Analogous to WORD. Here the inputs may be either words or sentences and the value is a sentence.

SENTENCE OF "MO" AND "ZART" is "MO ZART"
SENTENCE OF "AB" AND "CD EF" is "AB CD EF"
SENTENCE OF "" AND "APPLE" is "APPLE"

In the last example the output is a one-word sentence again.

7. COUNT (one input)

The output of the expression is the number of letters in the input if it is a word or the number of words if it is a sentence.

COUNT OF "ABC" is "3"
COUNT OF "THE CAT IN THE HAT" is "5"
COUNT OF "" is "0"

8. SUM (two inputs)

Both inputs must be numbers (i.e., words consisting only of digits preceded by an optional + or - sign). The output of the expression is the signed sum of the two inputs, prefixed by a - sign if the sum is negative.

SUM OF "-5" AND "3" is "-2"
SUM OF "5" AND "-2" is "3"

9. DIFFERENCE (two inputs)

Analogous to SUM. The output is the result of subtracting the second input from the first.

DIFFERENCE OF "3" AND "-5" is "8"

10. MAXIMUM (two inputs)

Analogous to SUM. The output is the larger of the two inputs.

MAXIMUM OF 2 AND 4 is 4.

Integers do not need to be quoted in LOGO.

11. MINIMUM (two inputs)

Analogous to SUM. The output is the smaller of the two inputs.

12. RANDOM (no inputs)

The output is a digit between 0 and 9 generated in a pseudo-random manner. Larger pseudo-random numbers are generated by concatenation. Thus,

WORD OF RANDOM AND RANDOM, yields a random number between 00 and 99.

13. DATE (no inputs)

The output is the current date, a word representing month, day, year. For example, "10/31/1969".

14. TIME (no inputs)

The output is the current time, a sentence like "1:32 AM".

15. CLOCK (no inputs)

The output is a number giving the number of seconds elapsed since an internal clock was reset.

```
←RESET CLOCK
←  :   (some work taking about half an hour)
   :

←PRINT CLOCK
1836
←PRINT CLOCK
1838
←
```

16. REQUEST (no inputs)

When the computer evaluates the expression REQUEST, it pauses to allow the user to type in something (often a requested answer to a question) at the teletype. When the typing is completed (as indicated when the user types a carriage return), the output of the expression is the typed-in text. The following procedure shows the use of REQUEST.

```
←TO COPYCAT
>10 PRINT "TELL ME SOMETHING."
>20 PRINT REQUEST
>30 COPYCAT
>END
COPYCAT DEFINED
←

←COPYCAT
TELL ME SOMETHING.
*WHO ARE YOU?
WHO ARE YOU?
TELL ME SOMETHING.
*WHY SHOULD I?
WHY SHOULD I?
TELL ME SOMETHING.
*ARE YOU SOME KIND OF NUT
ARE YOU SOME KIND OF NUT
TELL ME SOMETHING.
*

:   :
:   :
```

The asterisk (*) is typed by the REQUEST command to indicate to the user that the computer is waiting for his typing.

17. ASK (one input)

This is similar to REQUEST except that there is an input - the maximum number of seconds the computer should wait for type-in to be completed. If time runs out, the output of the expression is the empty word.

18. THING (one input)

The output of this expression is the thing named by the input. THING OF "X" is exactly the same as /X/. The utility of THING lies in expressions like THING OF /X/ (the analogous //X// is illegal) and THING OF WORD OF /X/ AND /Y/.

The following are all predicates; i.e., they output TRUE or FALSE.

19. IS (two inputs)

This is the most general of the built-in predicates. Most others could be built out of it. The output of the expression is "TRUE" if the things expressed by the two inputs are identical, letter for letter; otherwise its output is "FALSE".

IS "CAT" "CAT" is "TRUE"
IS 03 3 is "FALSE"
IS LAST OF 03 FIRST OF 3 is "TRUE"

20. EMPTYP (one input)

Its output is "TRUE" if the input is the empty word. It is "FALSE" otherwise.

EMPTYP OF /X/ has the same effect as
IS /X/ "" or
IS /X/ /EMPTY/

21. ZEROP (one input)

The input must express a number, otherwise there is an error. If the input is equal to 0 (+0, -0, 00, etc.), the output of the expression is "TRUE". If the input is a non-zero number, the output of the expression is "FALSE".

22. WORDP (one input)

WORDP outputs "TRUE" if its input is a word (not a sentence). It outputs "FALSE" otherwise.

23. SENTENCEP (one input)

Like WORDP, except it outputs "TRUE" if the input is a sentence. SENTENCEP and WORDP both output "TRUE" for the empty word. For any other input their outputs are opposite.

24. NUMBERP (one input)

NUMBERP outputs "TRUE" if its input is a number in standard form (that is, 123, +17, -000 give "TRUE", while A37, 7+8, ++3, 7.5 give "FALSE"). NUMBERP outputs "TRUE" for precisely those things which are legal inputs for SUM, DIFFERENCE, ZEROP, GREATERP, MAXIMUM, and MINIMUM.

25. GREATERP (two inputs)

The inputs must be numbers. GREATERP outputs "TRUE" if the first input is larger than the second. It outputs "FALSE" if the first is less than or equal to the second.

1.8 List of Elementary Commands

1. TO

This command indicates the beginning of a procedure definition. Immediately following the TO on the same instruction line is the name of the procedure being defined (this must be a word, not a sentence) and the names of its inputs, if it has any.

2. END (no inputs)

This indicates the completion of a procedure definition.

3. OUTPUT (one input)

This command causes a procedure to output the LOGO word or sentence specified in its input. It can only be used within the definition of a procedure. When the procedure is running and the OUTPUT command is encountered, its input becomes the output of the procedure. The procedure then stops and LOGO proceeds with its program by running the instruction that called this procedure.

4. STOP (no input)

Like the command OUTPUT, STOP causes a procedure to stop (but without causing it to output). Again, as with OUTPUT, the program then goes on with the instruction that invoked this procedure.

5. GO TO LINE (one input)

Only used within the definition of a procedure. The input must be the number of a line in that procedure (or an operation whose output is such a number). When the procedure runs, execution of the GO TO LINE command causes the computer to execute its next instructions in numerical sequence beginning with the line referred to in the command's input (instead of continuing in its current numerical sequence).

6. LOCAL (one input)

Only used within a procedure definition. The command causes its input to become a local name as in the case with procedure inputs. (See Section 1.6 for detailed discussion.)

7. TEST (one input)

The input must either be one of the two words "TRUE" or "FALSE" or an operation which outputs one of them. The result of the command is to set the "truth flag" either to true or false. The "truth flag" is automatically local to every procedure and is initially set to true.

8. IF TRUE (one input)

Here the input may be a command or an operation. The status of the truth flag is tested and the input is executed if the flag is true.

9. IF FALSE (one input)

Like IF TRUE, except that its input is executed if the flag is false.

10. GOODBYE (no inputs)

Disconnects the student from the computer and turns off his teletype.

11. PRINT (one input)

Causes the input to be typed on the teletype followed by a carriage return - line feed.

12. TYPE (one input)

Like PRINT but without the final carriage return - line feed. TYPE facilitates the typing of a series of printouts on a single line.

13. MAKE (two inputs)

The first input becomes the name of the second input, as discussed in detail in Section 1.6.

14. DO (one input)

The input must be a LOGO instruction. The DO command causes this instruction to be executed.

15. RESET CLOCK (no inputs)

Causes a special LOGO one-second counter, CLOCK, to be reset to zero. CLOCK is started off at zero when a user starts up LOGO. It is incremented automatically.

16. WAIT (one input)

The input must be a number. The command causes the computer to pause that number of seconds. Pauses of more than 24 hours are illegal.

2. The LOGO System

We distinguish the LOGO system from the LOGO language as follows. The language consists of all those things (the operations, commands, names, etc., and the rules governing their relations and usage) necessary to express an executable LOGO program. The system consists of those additional things - features and facilities - that aid a user in his programming work at the computer terminal. These have to do mainly with program manipulation and debugging capabilities such as listing, editing, storing, and retrieving.

2.1 Editing

After a procedure has been defined and run, it often becomes necessary to make some changes in its definition. This can be done using the command EDIT. To illustrate the use of EDIT, consider the following definition of the procedure REVERSE.

```
←TO REVERSE /Y/
>10 TEST EMPTYP OF /X/
>20 OUTPUT WORD OF LAST OF /X/ AND REVERS OF BUTLAST OF /X/
>END
REVERSE DEFINED
←
```

There are three errors in this definition. First, a line is needed between 10 and 20 telling what to do if /X/ is the empty word. That can be fixed by the following instructions.

```
←EDIT REVERSE
>15 IF TRUE OUTPUT /EMPTY/
```

The first instruction, using the EDIT command, tells LOGO that the definition of REVERSE will be modified. The second instruction defines a new line in the procedure. This line is inserted as number 15 between lines 10 and 20. (Here you see our reason for generally numbering lines 10, 20, 30, ... instead of 1, 2, 3, ... - to leave room for subsequent insertions.)

The second error is a bug in the title line. There the input is referred to as /Y/ but elsewhere in the procedure as /X/. The title line is changed as follows.

```
>TITLE TO REVERSE /X/
>
```

Last, in line 20 REVERSE is spelled without the final E. We can correct this by simply retyping the line.

```
>20 OUTPUT WORD OF LAST OF /X/ AND REVERSE OF BUTLAST OF /X/
```

Now we have finished fixing the procedure, so we type END.

```
>END
REVERSE DEFINED
```

After LOGO acknowledges the redefinition of REVERSE, we can try out the modified procedure.

```
←PRINT REVERSE OF "PITH"
HTIP
←
```

There is a useful feature which could have reduced our work in correcting line 20. The command EDIT LINE (one input) tells LOGO that the user wants to make changes in the line specified. In order to avoid retyping of correct words in the old line being corrected, the computer recognizes the key N^c (indicating the joint striking of the control key and the letter N key on the teletype) as representing "the next word in the old line". Each time N^c is struck, it causes the next word of the old line to be typed. Thus, in our example (the user's typing is underscored):

```
>EDIT LINE 20
>20 N^cOUTPUT N^cWORD N^cOF N^cLAST N^cOF N^c/X/ N^cAND N^cREVERS \E
    R^cOF BUTLAST OF /X/
>
```

(Since N^c and R^c don't type out anything on the teletype, the above line looks readable.) The R^c (standing for the rest or remainder of the old line) indicates to LOGO that it is to provide enough N^c's to finish the line. The backslash (\) before the E is used to erase the space the computer typed after REVERS. (In general, the backslash character \ erases the preceding character, \\ erases the two preceding characters, and so on.) W^c erases the preceding word (back to the first space) and types a \ for every character it erases. Backslash and W^c work during all typing, not just during editing.

2.2 Abbreviating

To reduce the user's typing, the computer recognizes short forms for most commands. These are called abbreviations.

```
←P S OF "CAT" AND "DOG"
CAT DOG
```

P is the abbreviation for PRINT and S for SENTENCE. The long forms are substituted internally for the abbreviations as soon as the abbreviations are typed in. Thus, if you were to type in a procedure definition using abbreviations and then list it, the computer would type it back to you in expanded form.

Also, text included between quotation marks or slashes is not interpreted as an abbreviation. Thus,

```
←P "P P P S"
P P P S
←
```

The student can make his own abbreviations with the command ABBREVIATE (two inputs). The first input can be any LOGO thing. The second must be a word which will become the abbreviation.

```
←ABBREVIATE "PRINT SUM" "+"
←+ "3" "5"
8
←
```

2.3 Listing and Erasing

The command LIST causes the computer to type out, in standard format, the entity or entities specified by its input. The command has several forms.

1. LIST (one input)

The input here must be a procedure name. The computer types the definition of the procedure.

```
←LIST REVERSE

TO REVERSE /X/
10 TEST EMPTYP /X/
20 IF TRUE OUTPUT /EMPTY/
30 OUTPUT WORD OF LAST OF /X/ AND REVERSE OF BUTLAST OF /X/
END
←
```

2. LIST ALL PROCEDURES

The computer lists all the procedure definitions currently in the student's workspace (see Section 2.5).

3. LIST CONTENTS

Lists the title line of every defined procedure currently in the student's workspace.

```
←LIST CONTENTS

TO REVERSE /X/
TO PRODUCT /X/ AND /Y/
TO FACTORIAL /N/
←
```

4. LIST ALL NAMES

All names whose things are not the empty word are listed.

```
←LIST ALL NAMES

/X/ IS "OLD THING"
/CAT/ IS "HOTDOG"
/N/ IS "15"
←
```

5. LIST ALL ABBREVIATIONS

All student-defined abbreviations are listed.

```
←LIST ALL ABBREVIATIONS

R:    REVERSE
PR:   PRODUCT
!:    FACTORIAL
+:    PRINT SUM
←
```

6. LIST ALL

All procedures, all names, and all abbreviations are listed.

The following two list instructions have meaning only while a procedure is being defined or edited.

7. LIST TITLE

The title line of the procedure is listed.

8. LIST LINE (one input)

The input must be a number. That line is listed.

- - -

The command ERASE provides a means of removing material from the computer's memory. The forms of the ERASE command are similar to those for LIST.

1. ERASE (one input)

The input must be a procedure name, as with LIST. That procedure definition is erased.

2. ERASE ALL PROCEDURES

All procedure definitions currently in the student's workspace are erased.

3. ERASE ALL NAMES

All names are given empty things.

4. ERASE ALL ABBREVIATIONS

All abbreviations are forgotten.

5. ERASE ABBREVIATION (one input)

Just that specific abbreviation is erased.

```
←ERASE ABBREVIATION "+"
←
```

+ would no longer be an abbreviation for PRINT SUM.

6. ERASE ALL

The computer is restored to its initial state, as it was when the student first entered.

The following instruction is used only while defining or editing a procedure.

7. ERASE LINE (one input)

The input must be a number. The indicated line is deleted from the procedure definition.

Two special commands indirectly involve listing. The command BURY makes a procedure unlistable. This command can only be used by a teacher (the computer recognizes a teacher by his password). It has proved useful in presenting assignments. The teacher writes a procedure, buries it, and then asks the students to write a procedure that has the same effect as the buried one. DIGUP (also for use of the teacher only) undoes the effect of BURY.

2.4 Debugging

The LOGO system has built-in aids to help students find the bugs in their programs. A bug will have one of two effects. It may cause the computer to try to execute an illegal instruction or it may direct the execution of instructions that are legal but which produce a wrong answer or no answer at all, e.g., it may put the computer in a loop that never ends.

In the first case, the computer immediately stops doing instructions and types out a diagnostic message describing the error and telling where it occurred. (Some typical diagnostic messages are listed at the end of this section.) Here is an example of the first kind of bug. Let us define a procedure GREET.

```
←TO GREET /X/
>10 PRINT SENTENCE OF "HELLO," AND /X/
>20 PRONT "HOW ARE YOU?"
>30 PRINT "SEE YOU LATER"
>END
GREET DEFINED
```

Now let's run it.

```
←GREET "JOHN"
HELLO, JOHN

PRONT NEEDS A MEANING.
I WAS AT LINE 20 IN GREET
```

There was a bug. The diagnostic message tells us what is wrong and where the error was found. So we fix the bug.

```
←EDIT GREET
>20 PRINT "HOW ARE YOU"
>END
GREET DEFINED
```

We try again.

```
←GREET "JOHN"
HELLO, JOHN
HOW ARE YOU?
SEE YOU LATER
←
```

This time GREET works.

When the procedure GREET was being defined, the computer didn't object when line 20 was typed in, nor should it have. It is possible that a procedure PRONT might have been written later, after GREET was defined. And, if the student had defined PRONT, before running GREET, for example -

```
←TO PRONT /X/
>10 PRINT /X/
>END
PRONT DEFINED
←
```

GREET would have worked perfectly well.

In the above example, the computer's diagnostic message pointed to the source of the error and thus was directly helpful. Often, however, we get situations where the illegal instruction isn't the cause of the error at all. For example, in the course of running a procedure the computer may say

```
DIFFERENCE OF "AB" AND "1"?  INPUTS MUST BE NUMBERS.
I WAS AT LINE 30 OF PRODUCT.
```

And when we look at PRODUCT we see something like

```
30 OUTPUT SUM OF /X/ AND PRODUCT OF /X/ AND DIFFERENCE OF
   /Y/ AND "1"
```

The error is that somehow /Y/ must have become "AB" instead of a number. But, the location of the error isn't line 30. /Y/ is being set up incorrectly somewhere else. This type of error then is really like the second kind mentioned above. The computer gets past it without performing an illegal instruction but it produces a result which shows up as faulty later when the computer is performing another instruction, perhaps in a different procedure. In this case the diagnostic is less helpful and more work must be done to find the error.

The most powerful method for pinpointing errors of this sort is to plant, at strategic spots in the procedures, lines of the form PRINT SENTENCE OF "AT LINE --- IN PROCEDURE --- /Y/ IS" AND /Y/ with the blanks filled in appropriately. Now, when the procedures run, they will leave a trace showing the things of "Y" and how they change. Using this trace, it is easy to see where /Y/ goes wrong. After the bug is fixed, the tracing lines can be erased. To reduce the editing work required to put in and subsequently remove tracing lines, the LOGO system has the built-in facility of tracing title lines and output lines. The operation of the TRACE command is illustrated in the following example, another REVERSE procedure.

```
←TO REVERSE /X/ AND /Y/    (/Y/ should start as the empty word)
>10 TEST EMPTYP /X/
>20 IF TRUE OUTPUT /Y/
>30 OUTPUT REVERSE OF BUTLAST OF /X/ AND WORD OF /Y/ AND
    LAST OF /X/
>END
REVERSE DEFINED
```

This is a correct procedure.

```
←PRINT REVERSE OF "CAT" AND ""
TAC
```

This is how we put a TRACE on it.

```
←TRACE REVERSE
```

This is what happens when we run a traced procedure.

```
←PRINT REVERSE OF "CAT" AND ""
REVERSE OF "CAT" AND ""
  REVERSE OF "CA" AND "T"
    REVERSE OF "C" AND "TA"
      REVERSE OF "" AND "TAC"
      REVERSE OUTPUTS "TAC"
    REVERSE OUTPUTS "TAC"
  REVERSE OUTPUTS "TAC"
REVERSE OUTPUTS "TAC"
TAC
```

To remove the TRACE on REVERSE we simply write

```
←ERASE TRACE REVERSE
←
```

The LOGO commands TRACE ALL PROCEDURES and ERASE ALL TRACES are useful with programs involving several procedures, and particularly with recursively chained procedures.

Diagnostic Messages

There are about 100 diagnostic messages. The following are some typical ones.

```
THAT ISN'T YOUR FILE.
MEANINGLESS CHARACTER.
IF WHAT?  (IF TRUE OR IF FALSE ONLY).
YOU NEED / MARKS AROUND EACH INPUT.
THE TITLE MUST BEGIN WITH TO.
END WHAT?  YOU'RE NOT DEFINING ANYTHING.
GO WHERE?
LIST WHAT?
```

```
ERASE WHAT?
YOU CAN'T TRACE BUILT-IN OPERATIONS.
DON'T USE THE EMPTY WORD FOR A NAME.
THE INPUTS TO WORD MAY NOT BE SENTENCES.
ILLEGAL COMMAND.
THE INPUT TO TEST MUST BE A PREDICATE.
YOU FORGOT THE LINE NUMBER.
```

The following four comments mean that the number of inputs found on the line and the number needed didn't match. The exact comment chosen depends on the particular parsing error.

```
SOMETHING EXTRA
SOMETHING MISSING
SOMETHING EXTRA IN A NAME        (with the MAKE command)
SOMETHING EXTRA IN A THING          "    "    "      "
```

In the following diagnostics, the underscored words are filled in appropriately by LOGO when the error occurs. The words given here are typical examples.

```
MATCHING "?  (or / or ( or ))
PRONT NEEDS A MEANING.
TRUMP ISN'T COMPLETELY DEFINED.  (END command not yet given.)
THERE IS NO LINE 30.
SUM OF "A" AND "5"?  INPUTS MUST BE NUMBERS.
    (Similar comments for DIFFERENCE, MAXIMUM, MINIMUM, GREATERP,
     ZEROP, ASK, and WAIT)
TEST IS USED BY LOGO.  CHOOSE ANOTHER PROCEDURE NAME.
    (The student can't define a procedure called TEST)
OF ISN'T A PROCEDURE.
THERE ISN'T ANY FILE GRANT
ANAGRAM ISN'T IN THAT FILE.
REVERSE ISN'T TRACED.
REVERSE IS ALREADY TRACED.
REVERSE IS ALREADY DEFINED.
YOU'RE ALREADY DEFINING REVERSE.
YOU'RE ALREADY EDITING REVERSE.
REVERSE STOPPED WITHOUT AN OUTPUT.  IT CAN'T BE USED AS AN INPUT.
```

The comment I AM IN TROUBLE. TELL YOUR TEACHER indicates a computer failure.

2.5 Filing

An important aspect of writing programs in LOGO is building complex programs from simpler ones. For example, assume a MULTIPLY procedure has been written. Sometime later the student may write a FACTORIAL procedure using the MULTIPLY. Then, perhaps, a PROBABILITY procedure using FACTORIAL and other procedures. Finally, PROBABILITY might end up in some game-playing strategy program.

LOGO contains a facility for filing away procedure definitions. The basic unit of a LOGO file is an entry. This is like a single file folder and may contain procedure definitions, names, and abbreviations. In a well organized file, each entry contains a related group of procedures, names, and abbreviations (for example, those that are used for playing NIM, or those used in solving linear equations).

An entry has a name which consists of two words. The first word is the file name and is common to all the entries in a file (it is often the name of the student who owns the file). The second word usually describes the entry and distinguishes it from other entries in the same file. Examples of names are JIM EQUATIONS, NANCY RANDOMSENT, SCOTT NIM.

An entry is created by the command SAVE. The entry contains everything that would be listed by LIST ALL, that is, all procedures, all names, and all abbreviations made by the student during this session. This material, comprising everything in his active area of the computer's memory, is called the student's workspace.

```
←SAVE GRANT ARITH (GENERAL ARITHMETIC FUNCTION)
←
```

In the example above the entry GRANT ARITH would be created (if GRANT ARITH already existed, the old entry would be erased and replaced by the new one). The entire workspace would be copied into the entry. There would then be two copies of the workspace, one in the entry and one still actively in the computer where the student is using it. The active copy is erased when the student gives the command GOODBYE.

The parenthesized text "GENERAL ARITHMETIC FUNCTION" is saved with the entry as a comment.

```
←LIST FILE GRANT

ARITH    (GENERAL ARITHMETIC FUNCTION 9:44 AM 10/1/1969)
REVERSE  (WRITES ITS OWN PROCEDURES 3:37 PM 9/8/1969)
BINTEST  (ADDTEST AND MULTEST IN BINARY 2:37 PM 8/2L/1969)
←
```

Also in the comment is the time and date that the entry was saved. Indeed, if no comment is given, as in the following case,

```
←SAVE GRANT SQUARE-ROOT
←
```

a comment containing only the time and date of saving is constructed and saved with the entry.

To retrieve an entry from a file, the command GET is used.

```
←GET GRANT ARITH
(GENERAL ARITHMETIC FUNCTION 9:44 AM 10/1/1969)
←
```

The associated entry comment is typed out and the contents are copied into, and become a part of, the student's active area (workspace). It is important to note that the entry itself is not removed from the file - what is placed in the student's workspace is merely a copy. Thus, any number of students can get a single entry at the same time.

To remove an entry from a file, the command ERASE ENTRY is used:

```
←ERASE ENTRY GRANT ARITH
←
```

When all entries in a file are erased, the file itself is automatically eliminated.

A file is created by creating its first entry. The file is owned by the student who created it. Only he may SAVE or ERASE entries in that file. Anyone can GET or LIST from it, however, unless the owner makes some entries private.

The command LOCK is used to make an entry private. A student can only LOCK entries in his own files. Thus,

```
←LOCK GRANT REVERSE
←
```

For anyone except the owner of the file GRANT, it will now be as if the entry GRANT REVERSE didn't exist. The command UNLOCK removes the lock on an entry.

```
←UNLOCK GRANT REVERSE
←
```

There are several forms of the LIST command that use files. (Underscores indicate places for appropriate names.)

LIST ALL FILES Types each file name.

LIST FILE _____ Types the name and comment for each entry in the specified file (as in the example above).

LIST ENTRY _____ _____ Types the comment, all the procedures, all the names, and all the abbreviations in the entry. The format is like that of LIST ALL.

LIST COMMENT _____ _____ Types the comment for that entry.

LIST PROCEDURES _____ _____ Types the definitions of all procedures in the entry.

LIST NAMES _____ _____ Types all names in the entry (and their things).

LIST ABBREVIATIONS _____ _____ Types all abbreviations in the entry.

LIST CONTENTS _____ _____ Types only the title lines of all procedures in the entry.

3. Summary of LOGO Operations, Commands, Special Names, and Abbreviations

OPERATIONS

FIRST	(1 INPUT) FIRST LETTER OF WORD OR FIRST WORD OF SENTENCE
LAST	(1 INPUT) LAST LETTER OF WORD OR LAST WORD OF SENTENCE
BUTFIRST	(1 INPUT) ALL BUT THE FIRST
BUTLAST	(1 INPUT) ALL BUT THE LAST
WORD	(2 INPUTS) CONCATENATES THE TWO INPUTS INTO A WORD
SENTENCE	(2 INPUTS) CONCATENATES THE TWO INPUTS WITH A SPACE BETWEEN THEM
COUNT	(1 INPUT) THE NUMBER OF LETTERS IN A WORD OR WORDS IN A SENTENCE
SUM	(2 INPUTS) THE ALGEBRAIC SUM OF TWO INTEGERS
DIFFERENCE	(2 INPUTS) THE ALGEBRAIC DIFFERENCE OF TWO INTEGERS
MAXIMUM	(2 INPUTS) THE LARGER OF TWO INTEGERS
MINIMUM	(2 INPUTS) THE SMALLER OF TWO INTEGERS
EMPTYP	(1 INPUT) TRUE OR FALSE AS INPUT IS /EMPTY/ OR NOT
ZEROP	(1 INPUT) TRUE OR FALSE AS INPUT IS 0 OR NOT
WORDP	(1 INPUT) TRUE OR FALSE AS INPUT IS WORD OR NOT
SENTENCEP	(1 INPUT) TRUE OR FALSE AS INPUT IS SENTENCE OR NOT
NUMBERP	(1 INPUT) TRUE OR FALSE AS INPUT IS NUMBER OR NOT
BEFOREP	(2 INPUTS) TRUE OR FALSE AS FIRST INPUT IS BEFORE SECOND OR NOT (BOTH INPUTS MUST BE TIME AND DATES OR JUST TIMES OR JUST DATES)
GREATERP	(2 INPUTS) TRUE OR FALSE AS FIRST INPUT IS LARGER THAN SECOND OR NOT (BOTH INPUTS MUST BE INTEGERS)
IS	(2 INPUTS) TRUE OR FALSE AS FIRST INPUT IS THE SAME AS SECOND OR NOT
THING	(1 INPUT) THAT THING THAT THE INPUT IS THE NAME OF
REQUEST	(NO INPUT) LITERAL TYPEIN FROM THE TELETYPE
ASK	(1 INPUT) LITERAL TYPEIN FROM THE TELETYPE IF COMPLETED IN INPUT NUMBER OF SECONDS. OTHERWISE /EMPTY/
RANDOM	(NO INPUTS) A RANDOM DIGIT
DATE	(NO INPUTS) THE CURRENT DATE
TIME	(NO INPUTS) THE CURRENT TIME
CLOCK	(NO INPUTS) A ONE SECOND CLOCK
BOTH	(2 INPUTS) LOGICAL AND OF TWO PREDICATES
EITHER	(2 INPUTS) INCLUSIVE OR OF TWO PREDICATES
ENTRIES	(1 INPUT) INPUT IS A FILE NAME. OUTPUT IS SENTENCE OF SECOND NAMES OF THE ENTRIES IN THAT FILE
DATE-SAVED	(1 INPUT) INPUT IS ENTRY NAME. OUTPUT IS TIME AND DATE ENTRY WAS SAVED
DATE-GOTTEN	(1 INPUT) LIKE DATE-SAVED
OWNER	(1 INPUT) INPUT IS A FILE NAME. OUTPUT IS NAME OF OWNER
INITIALS	(1 INPUT) INPUT IS ENTRY NAME. OUTPUT IS INITIALS OF SAVER
SIZE	(1 INPUT) INPUT IS ENTRY NAME. OUTPUT IS NUMBER DIRECTLY RELATED TO SIZE OF ENTRY ON THE DRUM

COMMANDS

TO	DEFINE PROCEDURE
TITLE	TO CHANGE TITLE LINE OF A PROCEDURE WHILE IN EDIT MODE
WAIT	TAKES ONE INPUT AND WAITS THAT MANY SECONDS
IF	FOLLOWED BY TRUE OR FALSE
EDIT	TO CHANGE A PROCEDURE
END	TO END A PROCEDURE DEFINITION
TRACE	CAUSES PRINTOUT WHILE A PROCEDURE RUNS
ERASE	ERASES MANY DIFFERENT THINGS
	1. ERASE ENTRY (ENTRY NAME)
	2. ERASE (PROCEDURE NAME)
	3. ERASE TRACE (PROCEDURE NAME)
	4. ERASE ALL TRACES
	5. ERASE ALL NAMES
	6. ERASE ABBREVIATION EXPRESSION
	7. ERASE ALL ABBREVIATIONS
	8. ERASE ALL PROCEDURES
	9. ERASE ALL
	10. ERASE LINE (NUMBER) (ONLY IN EDIT MODE)
LIST	LIST MANY THINGS
	1. LIST (PROCEDURE NAME)
	2. LIST ALL PROCEDURES
	3. LIST CONTENTS
	4. LIST CONTENTS (ENTRY NAME)
	5. LIST ALL NAMES
	6. LIST NAMES (ENTRY NAME)
	7. LIST ALL ABBREVIATIONS
	8. LIST ABBREVIATIONS (ENTRY NAME)
	9. LIST ALL
	10. LIST ALL FILES
	11. LIST FILE (FILE NAME)
	12. LIST ENTRY (ENTRY NAME)
	13. LIST TITLE (ONLY IN EDIT MODE)
	14. LIST LINE (NUMBER) (ONLY IN EDIT MODE)
GOODBYE	HALTS LOGO
PRINT	TYPES ITS INPUT AND THEN CARRIAGE RETURNS
TYPE	LIKE PRINT BUT WITHOUT CARRIAGE RETURN
OUTPUT	PROCEDURE ENDS AND HAS VALUE OF EXPRESSION FOLLOWING THE COMMAND
MAKE	SETS UP NAMES
STOP	PROCEDURE ENDS AND HAS NO VALUE

COMMANDS (continued)

DO	EXECUTES ITS INPUT AS A LOGO COMMAND
LOCAL	DECLARES FOLLOWING NAMES AS BELONGING TO PROCEDURE IN WHICH THE COMMAND IS
SAVE	SETS UP AN ENTRY
GET	READS IN AN ENTRY
GO	GO TO LINE ..
RESET	RESET CLOCK SETS CLOCK TO ZERO
ABBREVIATE	SETS UP ABBREVIATIONS
TEST	SETS TRUTH FLAG
PASSWORD	RESETS PASSWORD AND FILE DIRECTORY
LOCK	MAKES AN ENTRY PRIVATE
UNLOCK	UNDOES LOCK
HOARD	MAKES AN ENTRY NON-RESAVEABLE (WHEEL ONLY)
SHARE	UNDOES HOARD (WHEEL ONLY)
BURY	MAKES A PROCEDURE UNLISTABLE (WHEEL ONLY)
DIGUP	UNDOES BURY (WHEEL ONLY)

NAMES

/EMPTY/	THE EMPTY THING
/CONTENTS/	A SENTENCE OF DEFINED PROCEDURE NAMES
/LINE FEED/	A LINE FEED WITHOUT CARRIAGE RETURN
/CARRIAGE RETURN/	A CARRIAGE RETURN WITHOUT LINE FEED
/FILES/	A SENTENCE OF FILE NAMES
/FORM FEED/	ON SOME TELETYPES MOVES PAPER TO A NEW PAGE WHEN TYPED
/BLANK/	A BLANK SPACE
/BELL/	A BELL
/QUOTE/	A QUOTE MARK
/SKIP/	A NEW LINE (CARRIAGE RETURN AND LINE FEED)

```
                    ABBREVIATIONS

ABB:    ABBREVIATION
ABBS:   ABBREVIATIONS
ABT:    ABBREVIATE
BF:     BUTFIRST
BL:     BUTLAST
BP:     BEFOREP
C:      COUNT
DIFF:   DIFFERENCE
D-G:    DATE-GOTTEN
D-S:    DATE-SAVED
EDL:    EDIT LINE
EDT:    EDIT TITLE
EE:     ERASE ENTRY
EP:     EMPTYP
ER:     ERASE
ERL:    ERASE LINE
F:      FIRST
GB:     GOODBYE
GP:     GREATERP
GTL:    GO TO LINE
L:      LAST
LC:     LIST CONTENTS
LE:     LIST ENTRY
LL:     LIST LINE
MAX:    MAXIMUM
MIN:    MINIMUM
NP:     NUMBERP
P:      PRINT
PRS:    PROCEDURES
RQ:     REQUEST
OP:     OUTPUT
S:      SENTENCE
SP:     SENTENCEP
T:      TEST
W:      WORD
WP:     WORDP
ZP:     ZEROP
IFT:    IF TRUE
IFF:    IF FALSE
RT:     OUTPUT
GQ:     GREATERP
SQ:     SENTENCEP
WQ:     WORDP
NQ:     NUMBERP
EI:     EITHER
B:      BOTH

←
```